Projektinhallinta ja laadunvarmistus Globaalissa ohjelmistonkehityksessä

Ville Helminen

Pro gradu –tutkielma

Tietojenkäsittelytieteen laitos
Tietojenkäsittelytiede

Joulukuu 2015

ITÄ-SUOMEN YLIOPISTO, Luonnontieteiden ja metsätieteiden tiedekunta, Kuopio
Tietojenkäsittelytieteen laitos
Tietojenkäsittelytiede

Opiskelija, Ville Helminen: Projektinhallinta ja laadunvarmistus Globaalissa ohjelmistonkehityksessä
Pro gradu –tutkielma, 50 s.
Pro gradu –tutkielman ohjaaja: FT Jarmo Ahonen
Joulukuu 2015

Tiivistelmä: Globaali ohjelmistonkehitys (GSD) tarkoittaa hajautettua kehitystä, jossa kehittävät tiimit, tai asiakas ja tuottaja ovat maantieteellisesti eri paikoissa. Maantieteelliset erot voivat olla hyvinkin pieniä, jolloin useimmiten puhutaan vain hajautetusta ohjelmistonkehityksestä (DSD). GSD tarjoaa yrityksille pääsyn niukkojen resurssien pariin ympäri Maailman sekä mahdollisuuden saada kustannussäästöjä kehityksessä. Edellä mainitut ovatkin yritysten suurimmat syyt GSD –projektien toteuttamiselle. GSD:n implementoiminen ei kuitenkaan ole millään tavalla ongelmatonta. Ajalliset, maantieteelliset ja kulttuurilliset erot vaikeuttavat GSD -projektien johtoa. Erot aiheuttavat vaikeuksia kommunikaatiossa, joka aiheuttaa suurimman osan varsinaisista ongelmista. GSD:ssä projektinhallinta ja laadunvalvonta vaativat toimivaa ja kokenutta johtoa, jotta projektit voivat onnistua. Kommunikaation järjestäminen onkin yksi johdon tärkeimmistä tehtävistä. GSD –projekteja on tehty käyttäen hyväksi perinteistä vesiputousmallia sekä ketteriä menetelmiä. Molemmissa malleissa on haasteensa ja puolensa. Valitun kehitysmallin ja strategian jälkeen, muutosten ja laadunhallinnan tulee olla nopeaa ja keskitettyä.

Avainsanat: Globaali ohjelmistonkehitys, hajautettu kehitys, laadunvalvonta, projektinhallinta, kommunikaatio.

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry, Kuopio
School of Computing
Computer Science

Student, Ville Helminen: Project management and quality assurance in global software development
Master's Thesis, 50 p.
Supervisor of the Master's Thesis: PhD Jarmo Ahonen
December 2015

Abstract: In Global Software Development (GSD) the customer and developers or the developing teams are geographically in different places. In case the distances are short, the term Distributed Software Development (DSD) is often used. GSD offers many possibilities to companies. The most commonly desired advantages are cost savings and access to skilled employees around the World. Implementing GSD, however, is not trouble-free. Differences in locations, cultures and time zones make it difficult to manage GSD projects. The differences cause distress in communication, which causes biggest part of the actual problems. In GSD, project and quality management require experienced leadership. Making sure that communication works is a primary task for the management. GSD projects have been done using traditional and agile development techniques. Both techniques have their pros and cons. After choosing the techniques and strategy, quality assurance methods and responding to changes need to be swift and concentrated.

Keywords: Global software development, distributed develompent, quality management, project management, communication.

Lyhenneluettelo

BSC	Balanced Score Card; SLA:n suorituskyvyn mittareita
CAM	Contract and Account Management; sopimusten ja asiakkuuksien hallinta
CFA	Corporate Framerwork Agreement; puitesopimus, jossa määritellään sopimuksen yleisiä ehtoja.
CMM	Capability Maturity Model: organisaatioiden kypsyysmalli
DSD	Distributed Software Development; hajautettu ohjelmistonkehitys
GSD	Global Software Development; globaali ohjelmistonkehitys
GSO	Global Software Outsourcing; globaalisti ulkoistettu ohjelmistonkehitys
IEEE	Institute of Electrical and Electronics Engineer; kansainvälinen tekniikan alan järjestö
KMS	Knowledge Management System; tiedonhallintajärjestelmä
ODC	Orthogonal Defect Classification: ortogonaalinen virheluokittelu
SLA	Service Level Agreement ; palvelusopimus, jossa määritellään palvelulle tietyt vaatimustasot.
SQA	Software Quality Assurance; ohjelmistojen laadunvarmistus
XP	Extreme Programming; ketterän ohjelmistokehityksen eräs metodologia

Sisällysluettelo

1 Johdanto

Globaali ohjelmistonkehitys, hajautettu ohjelmistonkehitys ja ohjelmistonkehityksen ulkoistus ovat käsitteitä, joita käytetään alan kirjallisuudessa hieman päällekkäin. Loppujen lopuksi kyse on kuitenkin siitä, että saman ohjelmiston kehitystä tehdään eri lokaatioissa. Globaalissa ohjelmistonkehityksessä voi olla kyse pelkästään myös siitä, että asiakas ja ohjelmiston toteuttava yritys ovat eri valtioissa.

GSD tarjoaa yritykselle monia hyötyjä, kuten esimerkiksi kustannussäästöjä ja pääsyä niukkojen henkilöstöresurssien pariin, mutta myös ongelmia, joista suurin osa ilmenee puutteellisen kommunikaation takia. Projektinhallinta ja laadun ylläpito hajautetussa kehityksessä on monimutkainen kokonaisuus, joka tarvitsee ansioitunutta ja kokenutta johtoa. Erityistä huomiota tulee kiinnittää rikkaan kommunikaatioympäristön rakentamiseen.

Tämän tutkielman kappaleessa 2, Globaali ohjelmistonkehitys, käydään tarkemmin kiinni kirjallisuudessa esiintyviin termeihin ja pyritään selvittämään mistä eri termeissä on kyse. Termien selvittämisen jälkeen esitellään GSD:n tarjoamia mahdollisia hyötyjä ja haittoja; millaisia hyötyjä yritykset hakevat ja millaista hyötyä hajauttamisesta oikeastaan on. Kappaleessa esitellään myös erilaisia tapoja GSD –projektien suorittamiseen sekä perehdytään hieman toimittajan valintaan.

Kappaleessa 3 käydään läpi laadunvalvonnan asioita ohjelmistonkehityksen ja erityisesti hajautetun ohjelmistonkehityksen näkökulmasta. Tarkastellaan mitä laatu on ja mitä erityispiirteitä laadunvalvonnalla ja ohjelmiston tarkastusprosessilla on GSD:n näkökulmasta. Kappaleessa käydään läpi myös virheluokittelumalleja, sekä niiden mahdollista käyttöä GSD:ssä.

GSD eri lähestymistapoja –nimellä olevassa kappaleessa numero 4, tutkitaan hiukan tarkemmin eroja yrityksen sisällä ja yritysten välillä tapahtuvassa hajautetussa kehityksessä. Samassa kappaleessa myös käydään läpi perinteisen vesiputousmallin ja ketterien menetelmien käyttöä hajautetussa ohjelmistonkehityksessä - niiden tarjoamia etuja sekä haasteita.

Kappale numero 5 tuo paremmin esiin kommunikaatio-ongelmia, jotka voivat johtua esimerkiksi kulttuurieroista ja organisaatioiden eroista. Kappale käsittelee organisaation ja henkilöstön lisäksi suhteiden ja kulttuurien johtamista, josta siirrytään vielä johtamisen omaan kappaleeseen 6. Kyseisessä kappaleessa käydään läpi johdon haasteita tietohallinnon, sopimusten hallinnan, suhteiden hallinnan ja IT-strategian hallinnan parissa. Kappale esittelee monimutkaisen vyyhdin, jonka aukaisemiseen tarvitaan kokenutta ja osaavaa johtoa. Viimeinen kappale 7, on yhteenveto tästä tutkielmasta.

2 Globaali ohjelmistonkehitys

Globaalilla ohjelmistonkehityksellä, jatkossa GSD (*Global Software Development*), voidaan tarkoittaa ul Haq et. al. (2011) mukaan sopimussuhdetta asiakkaan ja toimittajan organisaatioiden välillä, jossa asiakas ulkoistaa osan tai koko ohjelmiston kehityksen toimittajalle. GSD, jota kutsutaan joissain yhteyksissä myös *hajautetuksi ohjelmistonkehitykseksi*, on alun perin ulkoistamistapa, jossa eri maantieteellisissä paikoissa olevat monikulttuuriset kehitystiimit tekevät yhdessä saman ohjelmiston kehitystyötä (ul Haq et. al., 2011).

Termejä GSD, hajautettu ohjelmistonkehitys ja ulkoistus käytetään alan kirjallisuudessa hieman päällekkäin ja osittain jopa sekavasti. Khan K. et. al (2013) mukaan edellä mainitut ovat pääasiallisesti eri nimiä ohjelmistonkehitystekniikalle, jossa asiakas saa palveluita ulkopuoliselta yritykseltä, jota kutsutaan myyjäksi tai toimittajaksi. Globaalia ohjelmistonkehitystä tapahtuu kuitenkin paljon myös yritysten sisällä, jolloin termiä ulkoistus ei pidä käyttää. Esimerkiksi Windows Vistan kehitys oli globaalisti hajautettua, mutta koko projekti toteutettiin saman yrityksen sisällä (Bird et al., 2009).

Heeks et. al. (2001) käyttävätkin käsitettä ”*globaali ohjelmistonkehityksen ulkoistus”* GSO (*Global Software Outsourcing*), kun kyseessä on kyseessä on IT:n tai kehityksen ulkoistus maantieteellisesti eri paikassa olevalle yritykselle. Gopal et. al. (2002) taas puhuvat *ulkoistetusta hajautetusta kehityksestä* nimellä ”*offshore software development*”. Nisar & Hameed (2004) mielestä offshore software development taas voi tapahtua sekä yrityksen sisällä että toiselle yritykselle ulkoistettuna. Ainoastaan GSO:sta puhuttaessa onkin selvä, että kyseessä on toiselle yritykselle ulkoistettu palvelu. GSD:stä puhuttaessa asia ei ole kirjallisuudessa niin yksinkertainen.

Carmel & Agarval (2001) kirjoittavat artikkelissaan GSD:stä rinnastaen sen offshore –kehitykseen. Herbsleb & Moitra (2001) eivät tee omassa kirjoituksessaan suurta eroa hajautetun ulkoistetun kehityksen ja hajautetun sisäisen kehityksen välillä. Heidän mukaansa ongelmat sekä ulkoistetussa että firman sisällä tapahtuvassa globaalissa kehityksessä ovatkin hyvin samankaltaisia. Layman et. al. (2006) puhuvat sekä hajautetusta ohjelmistonkehityksestä DSD (*Distributed Software Development*) että

globaalista ohjelmistonkehityksestä, GSD:stä. Heidän mukaansa DSD on kehitystä, jossa saman yrityksen sisällä olevat tekijät on hajautettu eri maantieteellisiin paikkoihin. Ero voi olla vierekkäisistä rakennuksista eri maanosiin saakka. GSD on taas erikoistapaus DSD:stä, jolloin työntekijät on hajautettu eri maihin. GSD sisältää sekä yrityksen sisäisen että toisille yrityksille ulkoistetun ohjelmistokehityksen (Layman et. al 2006).

Koska ongelmat ja hyödyt sekä yrityksen sisällä että ulkoistetussa kehityksessä ovat hyvin samankaltaisia (Herbsleb & Moitra 2001), käytetään tässä pro gradussa GSD –termiä, kun on kyseessä kehityksestä, joka tapahtuu useammassa kuin yhdessä valtiossa. Termiä GSD käytetään, oli kyse sitten yrityksen sisäisestä tai ulkoistetusta ohjelmistonkehityksestä. Selvyyden vuoksi esimerkeissä tarkennetaan kehityksen tapaa, mikäli tämä on tarpeellista.

Globaalisti hajautettua ohjelmistonkehitystä on yritetty hyödyntää jo vuosikymmeniä. Tekniikalla haetaan tavallisia ulkoistamisen hyötyjä; alempia kustannuksia, toiminnan tehostamista ja resursseja, joita ei ole saatavilla lähellä (Herbsleb & Moitra, 2011). Vaikka GSD:tä käytetään paljon sen mahdollistamien hyötyjen vuoksi, ei sen käyttöön ottaminen ole ollenkaan ongelmatonta (ul Haq et al., 2011). Tämän kappaleen luvuissa pureudutaan tarkemmin GSD:n mahdollistamiin hyötyihin ja ongelmakohtiin.

2.1 Hyödyt

GSD tarjoaa monenlaisia potentiaalisia etuja, joista mahdollisuus kustannussäästöihin on yksi useimmin mainituista (Herbsleb & Moitra 2011, Khan, S.U. et al., 2010). Kustannusten vähentäminen on mahdollista pienentämällä esimerkiksi palkkakuluja (Damien et. al. 2003, Carmel &Agarval 2001). Kustannussäästöjen lisäksi GSD mahdollistaa, vaikkakaan ei ongelmattomasti, tilaisuuden palkata kyvykkäitä työntekijöitä ohjelmiston yhtäaikaiseen kehittämiseen eri puolilla maailmaa ja pääsyn lähemmäs erilaisia markkinoita ja loppukuluttajia (Conchúir et. al. 2009). Conchúir et. al. (2009) tekemän tutkimuksen mukaan kaikki oletetut hyödyt eivät kuitenkaan toteudu. Heidän mielestään esimerkiksi innovaatioiden tekeminen ja parhaiden käytäntöjen jakaminen ei parane siirryttäessä paikallisesta ohjelmistonkehityksestä

GSD:hen. Myös maantieteellisten aikaerojen käyttö ei tuo sellaista hyötyä, kuin teoriassa voisi kuvitella (Conchúir et. al. 2009).

2.1.1 Asiakkaan hakemat hyödyt

Herbsleb ja Moitra kirjoittivat jo 2001 vuosikymmeniä kestäneestä liiketoiminnan globalisoitumisen lisääntymisestä, joka vaikuttaa varsinkin ohjelmistoalaan (Herbsleb & Moitra, 2011). Ohjelmistojen merkitys eri liiketoiminta-alueilla on jatkuvasti kasvava ja niiden hyödyntäminen voi tarjota yrityksille huomattavaa kilpailuetua toisiin yrityksiin nähden. Ohjelmistonkehitystä tekevät yritykset ovatkin lähteneet havittelemaan toimintojensa tehostamista ulkoistamalla kehitystä. Herbslebin ja Moitran (2011) mukaan muun muassa seuraavat asiat ovat vaikuttaneet globaalin ohjelmistonkehityksen lisääntymiseen: mahdollisuus käyttää ympäri maailman jakaantuneita niukkoja resursseja, mahdollisuus hyödyntää markkinoiden läheisyyttä ja tekijöiden paikallistuntemusta, mahdollisuus käyttää aikaeroa hyödyksi ja nopeuttaa täten tuotteiden tuomista markkinoille. Perimmäisenä tarkoituksena ulkoistuksessa on kuitenkin vähentää kustannuksia ja hyödyntää maantieteellisesti eripaikkoihin jakautuneita resursseja (Herbsleb & Moitra, 2011). Khanin, Niazin ja Ahmadin (2010) mukaan suurin syy ohjelmistonkehityksen ulkoistamiseen on tuotantokustannusten minimointi. Muita syitä ovat esimerkiksi pääsy toimittajan huipputeknologian pariin ja mahdollisuus keskittyä organisaation omaan ydinliiketoimintaan. Resursseja on paljon tarjolla maailmalla ja tuotteella on suurempi mahdollisuus päästä globaaleille markkinoille nopeammin, kuin tavallisessa ohjelmistojenkehityksessä (Khan, S.U. et al., 2010).

2.1.2 Tuntemattomat hyödyt

Globaalin ohjelmistonkehityksen tarjoamien resurssi- ja kustannussäästöjen sekä nopeamman kehityksen lisäksi, on tutkittu myös niin sanottuja tuntemattomia etuja, joita GSD voi tarjota. Nämä tuntemattomat edut voidaan jakaa kolmeen ryhmään: Organisaation edut, tiimin edut ja prosessien/tehtävien edut (Ågerfalk et al., 2008).

Organisaation edut voidaan edelleen jakaa kahteen ryhmään: Jaetut parhaat käytännöt ja innovaatiot, sekä resurssien tehokkaampi jakaminen (Ågerfalk et al., 2008). Ebert & Neven (2001) mukaan erilaisilla kulttuurisilla taustoilla olevat työntekijät

tekevät aktiivisesti töitä yhdessä parantaakseen tuotetta. He innovoivat uusia tuotteita ja kehittävät samalla prosesseja. Työnteon yhteydessä myös parhaat käytännöt eri kulttuureista jakaantuvat projektiin osallistuville (Ebert & Neve, 2001). Resurssien tehokkaampi jakaminen on kytköksissä kustannussäästöjen hakemiseen maista, joissa työvoima on halvempaa. Yritykset voivat siirtää mahdollisuuksien mukaan kalliimpia resurssejaan muihin, esimerkiksi strategisimpiin, tehtäviin (Ågerfalk et al., 2008).

Tiimin edut voidaan jakaa kolmeen eri ryhmään: tehtävien parempi jako moduuleihin, koordinoinnin pienemmät kustannukset ja tiimien itsenäisyyden lisääntyminen (Ågerfalk et al., 2008). GSD:n luonteelle on hyvin ominaista, että tiimit jakavat työn useisiin erillisiin moduuleihin (Ebert & Neve, 2001). Näitä moduuleita voidaan kehittää samanaikaisesti eri lokaatioissa useiden tiimien kesken. Tehtävien jako osiin johtaa siihen, että moduulista vastaavilla osapuolilla on vastuu ja velvollisuus tietyn moduulin koko elinkaaresta. Tämä taas vähentää riippuvuussuhteita ja siten kustannuksia (Battin et al., 2001).

Kustannussäästöjä voi syntyä myös koordinoinnin tarpeen vähetessä, kun tiimit tai niiden jäsenet työskentelevät eri aikaan. Kyseessä on niin kutsuttu ”auringon seuraaminen”, jossa käytetään aikaeroa hyväksi. Työstä osan suorittanut henkilö/tiimi siirtää työpäivän päätteeksi kehityksen toiselle tiimille. Tapa on huomattu toimivaksi muun muassa vikojen korjaamisen yhteydessä (Espinosa & Carmel, 2003). Kun kaksi henkilöä/tiimiä ei työskentele saman ongelman parissa yhtä aikaa, koordinoinnin tarve vähenee ja kulut pienenevät (Ågerfalk et al., 2008).

Tiimien itsenäisyys lisääntyy jonkin verran maantieteellisten erojen takia. Tämä itsenäisyys mahdollistaa tiimien erilaisten kulttuurien säilymisen, joka osaltaan edesauttaa tiimien tekemän työn laadun säilymisen hyvänä. Itsenäisyys kasvattaa tiimien mahdollisuutta päätöksentekoon, joka osaltaan saattaa hankaloittaa kehitettävän ohjelmiston koordinointia. Erityistä huomiota tulee silloin kiinnittää koko ohjelmiston vaatimuksiin. Tiimien tulee olla selvillä kehitettävän ohjelmiston lopputarkoituksesta, visiosta, jotta laatu saadaan säilytettyä hyvänä (Gumm, 2006).

Prosessien hyödyt voidaan kategorisoida kolmeen luokkaan: virallinen kirjallinen viestintä, parantunut dokumentaatio ja selkeästi määritellyt prosessit (Ågerfalk et al.,

2008). Koska GSD vaatii, esimerkiksi aikaerojen takia, usein kirjallista kommunikaatiota, jää tällaisesta kommunikaatiosta seurattava jälki. Tämä tarjoaa mahdollisuuden siihen, että asioiden alkuperä ja syy on helpompi selvittää (Ågerfalk, 2004).

Paremman dokumentaation syyt ovat melko samankaltaiset kuin virallisessa kirjallisessa viestinnässä. Eri paikkoihin hajautetut tiimit keskittyvät enemmän dokumentointiin parantaakseen kommunikaatiota ja tehdäkseen itsensä ymmärretyksi (Delone et al., 2005). Dokumentaatio auttaa muita projektiin osallistuvia ymmärtämään toisaalla tehtyjä päätöksiä. Dokumentaation lisäksi prosessien määritelmät tehdään hajautetuissa hankkeissa tarkemmin kuin ei hajautetuissa (Ågerfalk et al., 2008).

2.2 Haasteita

Suurien ohjelmistohankkeiden ulkoistamisessa on omat hankaluutensa. Yksi suurimmista haasteista on kommunikaation hoitaminen (Damian et al., 2007, Holmström et al. 2006). Globaali ohjelmistonkehitysprosessi on monimutkainen kokonaisuus, jossa erityisesti ajalliset, maantieteelliset ja kulttuuriset etäisyydet aiheuttavat odottamattomia ongelmia (Holmström et al. 2006, Korkala et al. 2010). Beulenin ja Ribbersin (2002) mukaan juuri informaation hallinnalla, sopimuksilla ja sopimusten hallinnalla on keskeinen rooli GSD:n onnistumisessa.

Cusumano (2008) nostaa yhdeksi suurimmista haasteista ketterien menetelmien käytön globaalissa ohjelmistonkehityksessä. Iteratiivisen kehityssuunnitelman lisäksi iteratiivisen sopimuksen tekeminen asiakkaan kanssa voi olla hankalaa, jos mahdollista. Useimmat asiakkaat haluavat sopia kiinteän hinnan ohjelmistonkehitykselle, mikä ei ole paras vaihtoehto kehittäjälle.

2.2.1 Kommunikaatio

Herbslebin ja Moitran (2001) mukaan kommunikaation ja koordinoinnin puute on merkittävä tekijä, kun hajautetussa ohjelmistonkehitysprojektissa aiheutuu viivästymisiä. Heidän mukaansa ohjelmistonkehitysprojekti vaatii kahta toisiaan täydentävää kommunikaatiotapaa. Muodollisen kanavan kautta projektiin osallistuvat tietävät muun muassa kuka on vastuussa mistäkin. Muodollinen kommunikaatiotapa tarvitsee selkeät ja hyvin ymmärretyt kanavat, joiden kautta informaatio kulkee projektiin

osallistuville. Toinen, vapaamuotoinen kommunikaatiotapa, saattaa aiheuttaa yllättävän paljon ongelmia globaalissa ohjelmistonkehityksessä. Koska työntekijät eivät ole samassa paikassa fyysisesti, on vapaamuotoista kommunikaatiota paljon vähemmän. Muun muassa tauoilla käytävä keskustelu siitä, kuka tekee mitäkin ja missä vaiheessa projekti heidän osaltaan on, puuttuu. Tämän seurauksena pieniä ja isoja asioita voi jäädä pitkäksi aikaa huomaamatta, tai jopa tekemättä. (Herbsleb &Moitra 2001). Ivček ja Galinac (2008) tuovat omassa tutkimuksessaan esiin kommunikaation tärkeyden myös asiakkaan kanssa.

DeLone, Espinosa, Lee ja Carmel (2005) esittävät haastattelututkimuksessaan kulttuuriin, kieleen ja koordinointiin liittyviä kommunikaatiovaikeuksia. Kulttuuriset erot vaikuttavat siihen, miten projektiin osallistuvat suhtautuvat asioihin ja kertovat esimerkiksi projektin etenemisestä. Kieli voi aiheuttaa ongelmia, vaikka se olisikin yhteinen. Henkilö, joka puhuu englantia mainiosti, mutta ei äidinkielenään, saattaa helposti olla ymmärtämättä tekstin vivahteita. Tutkimuksen mukaan merkittävä osa tietojärjestelmien ulkoistuksessa tapahtuvista ongelmista johtuu jollain tavalla huonosta kommunikaatiosta. Tehokkaat tiimit käyttävätkin GSD:ssä prosesseja, jotka on suunniteltu erityisesti kommunikaatio-ongelmien voittamiseen. (DeLone et al. 2005).

2.2.2 Ajallinen ja maantieteellinen etäisyys

Jopa pieni aikaero saattaa aiheuttaa ongelmia hajautetussa ohjelmistonkehityksessä, mikäli sitä ei ole otettu huomioon (Espinosa & Carmel, 2003). Tunnin aikaero tiimien välillä saattaa lyhentää yhteistä työaikaa päivän alussa, lounasaikaan ja päivän lopussa yhteensä useilla tunneilla (Grinter et al. 1999). Työajat ja arkipäivät voivat olla erilaisia eri maissa, jolloin yhteinen työaika saattaa jäädä hyvinkin lyhyeksi (Espinosa & Carmel, 2003).

Maantieteellinen ja ajallinen etäisyys aiheuttaa haasteita projektin koordinoinnin ja kommunikoinnin järjestämisessä (Khan et al. 2013). Yksinkertainen esimerkki tällaisesta on fyysisen tapaamisen järjestämisen ongelmat. Khan et al. (2013) mukaan tiimien sijoittaminen maantieteellisesti eri paikkoihin lisää informaatiokuilua. Samalla väärinymmärrysten riski kasvaa sekä kehitystiimien että asiakkaan ja tiimien välillä. Fyysiset, ajalliset ja kulttuuriset etäisyydet ovat esteenä reaaliaikaisen informaation jakamiseen hajautetussa ohjelmistonkehityksessä (Sangwan & Ros, 2008), mikä

vaikeuttaa kehityksen koordinointia. Samalla linjalla ovat Carmel ja Agarval (2001), joiden mukaan välimatka kehittäjien välillä pahentaa koordinoinnin ja kontrolloinnin ongelmia. Etäisyys vaikuttaa suoraan kielteisesti viestintään, mikä puolestaan vähentää koordinoinnin tehokkuutta.

2.3 GSD prosessi

Monimutkaisten hajautettujen projektien johtaminen vaatii uudenlaisia lähestymistapoja kommunikointiin ja uusia tekniikoita projektien hallintaan (Evaristo &Scrudder 2000). Gummin (2006) mukaan nykyisten menetelmien ja välineiden soveltuvuutta GSD:ssä voidaan analysoida vain, jos ymmärrämme hajauttamisen syyt ohjelmistokehityksessä. Kirjallisuuskatsauksessaan Gumm (2006) huomasi tutkijoiden keskustelevan hajautetusta kehityksestä näkökulmista, joihin liittyivät läheisesti seuraavat kysymykset: ”Kuka tai mitä on hajautettu, millä tavalla ihmiset ja muut kokonaisuudet on hajautettu, mitkä ovat hajautuksen erityiset haasteet ja miten hajautuksen haasteet voidaan parhaiten ratkaista”.

Gumm (2006) kehitti lajittelun, jossa erilaisien ulottuvuuksien avulla voidaan tarkastella hajautusta itseään, eli mitä on hajautettu ja millä tavalla. Hän erotteli ohjelmistonkehityksen hajautuksessa neljä eri ulottuvuutta, jotka ovat

- Fyysinen (tai maantieteellinen) etäisyys
- Organisatorinen etäisyys
- Ajallinen etäisyys
- Sidosryhmien etäisyys

Fyysinen etäisyys kuvaa ihmisten ja kokonaisuuksien välisiä sijainteja. Etäisyys voi vaihdella saman rakennuksen eri kerroksista toisiin mantereisiin. Etäisyyden aiheuttavaa kommunikaation ja kehityksen vaikeutta voidaan vähentää käyttämällä esimerkiksi palveluita, jotka mahdollistavat lähdekoodin kehittämisen keskitetysti. Organisatorisessa etäisyydessä voi olla kyse yrityksen eri osastojen eroavaisuuksista tai toimittajan ja hankkijan kulttuurillisista eroista. Erot voivat johtaa siihen, että tiimin jäsenillä on erilaiset näkemykset kehitysprosessin prioriteeteista. Ajallisella etäisyydellä mitataan aikavyöhykkeiden aiheuttamaa eroa, jonka takia projektin työntekijöil-

lä on rajallinen aika kommunikoida reaaliaikaisesti toistensa kanssa. Gummin mukaan ajallisella etäisyydellä ei ole suurta negatiivista vaikutusta. Sidosryhmien etäisyydellä tarkoitetaan esimerkiksi sitä, miten vaatimukset ovat hajautuneet eri sidosryhmiin. Sidosryhmillä voi olla erilaisia vaatimuksia, jotka pitää saada keskitetysti projektiryhmälle. Tämä vaatii hyvää ja keskitettyä dokumentoinnin ja päätöksenteon hallintaa. (Gumm 2006).

2.3.1 Ketterät menetelmät

Kuten aiemmin tutkielmassa on jo todettu, kommunikaatio on hyvin tärkeässä roolissa globaalissa, hajautetussa ohjelmistonkehityksessä. Korkala, Pikkarainen ja Conboy kuvaavat julkaisussaan perinteisen ja ketterän ohjelmistonkehityksen eroja GSD:ssä (Korkala et al., 2010). Ketterät menetelmät korostavatkin juuri kommunikaation tärkeyttä asiakkaan kanssa (Korkala et al., 2010). Ympäristö, jossa kommunikaatio on monipuolista ja rikasta, auttaa pienentämään etäisyyksien aiheuttamia negatiivisia seikkoja. Tällaisen ympäristön rakentaminen on tärkeä tekijä ketteriä menetelmiä hyödyntäessä (Layman et al., 2006).

Ketterien menetelmien puolesta kirjoitetussa julistuksessa, ”Manifesto for Agile Software Development”, seitsemäntoista perinteisen ohjelmistokehityksen kyseenalaistajaa tiivistivät arvonsa ketteristä menetelmistä seuraaviin asioihin (http://www.agilemanifesto.org/):

- Yksilöt ja kanssakäyminen ovat tärkeämpiä kuin menetelmät ja työkalut
- Toimiva ohjelmisto on tärkeämpi kuin kattava dokumentaatio
- Asiakasyhteistyö on tärkeämpää kuin sopimusneuvottelut
- Muutoksiin vastaaminen on tärkeämpää kuin suunnitelmissa pitäytyminen

Edellä mainitut tavat ovat kumminkin hieman ristiriidassa GSD:n kanssa. Rameshin, Caon, Mohanin ja Xun mukaan hajautettu ohjelmistonkehitys nojaa tyypillisesti muodollisiin prosesseihin, kun taas ketterissä menetelmissä koordinoinnin helpottamiseen käytetään vapaampia käytäntöjä (Ramesh et. al., 2006). Ramesh et. al. (2006) mukaan hajautetussa ja ketterässä ohjelmistonkehityksessä on muitakin haasteita. Heidän mielestään kyse on tasapainon löytämisessä. Miten ihmissuuntautunut ja prosessisuuntautunut malli saadaan toimimaan keskenään, miten viralliset sopimukset ja

epämuodolliset sopimukset saadaan balanssiin ja miten kommunikointi kaikkien näiden asioiden kesken hoidetaan. Tutkimuksessa tullaan lopputulokseen, että ketterien menetelmien varovainen liittäminen GSD:hen on mahdollista ja kannattavaakin. GSD:n ja ketterien menetelmien yhdistämisen auttamiseksi asianosaisten tulee jatkuvasti kehittää prosessia, parantaa kommunikaatiota ja edesauttaa tiedon jakautumista sekä rakentaa luottamusta projektiin osallistuvien välillä (Ramesh et. al., 2006).

2.3.2 Kumppanin/toimittajan valinta

GSD:n avulla yritykset pyrkivät hakemaan muun muassa kustannussäästöjä ja laadukasta työvoimaa. Khanin, Niazin ja Ahmadin (2010) kirjallisuuskatsauksen mukaan asiakkaan näkökulmasta kuusi tärkeintä seikkaa, jotka vaikuttavat toimittajan houkuttelevuuteen, ovat:

1. Kustannussäästöt
2. Ammattitaitoiset työntekijät
3. Sopiva infrastruktuuri
4. Tuotteiden ja palveluiden laatu
5. Tehokas ulkoistettujen palveluiden hallinta
6. Referenssit onnistuneista projekteista

Khan et. al. (2010) mukaan ostajan kannattaa tarjoajaa valitessaan kiinnittää yllä olevista huomiota erityisesti pätevään työvoimaan, sopivaan infrastruktuuriin ja tuotteiden sekä palvelun laatuun. Näitä tekijöitä tukevat heidän mukaansa myös muut tutkimukset. Sopivalla infrastruktuurilla Khan, Niaz ja Ahmad (2010) tarkoittavat verkko- ja tietoliikenneinfrastruktuuria, fyysistä infrastruktuuria (rakennukset, tiet, vesi, sähkö ja niin edelleen) ja riittäviä resursseja ylläpitää isoja kehityshankkeita.

Toimittajan valintaprosessiin on olemassa erilaisia malleja. Aissaoui, Haoauri ja Hassini (2007) esittelevät tutkimuksessaan nelivaiheisen prosessin valintaa helpottamaan. Mallia ei ole kehitelty erityisesti GSD:tä varten, vaan yleisesti toimittajan valintaa helpottamaan. Mallin ensimmäinen vaihe on ongelman määrittely. Määrittelyn tuloksena saadaan selville mitä toimittajan valinnalla halutaan ratkaista. Toinen vaihe on niiden kriteerien määrittely, jolla toimittaja valitaan. Esimerkiksi laatu, hinta ja niin edelleen. Kolmannessa vaiheessa tehdään esivalinta mahdollisten toimittajien

kesken. Toimittajia muun muassa vertaillaan asetettuihin kriteereihin ja poistetaan sopimattomat. Neljännessä vaiheessa valitaan sopivin/sopivimmat toimittajat ja tehdään tilaus. (Aissaoui et al., 2007)

3 Laadunvalvonta

Hajautetun ohjelmistokehityksen myötä projektin johtamiseen kohdistuu uusia haasteita. Jotta ohjelmiston laatu voidaan pitää hyvänä, täytyy GSD:n käytäntöjä kehittää. Kehittämistä varten tarvitaan laadunvalvontaa (Ivček & Galinac, 2008). Khan et. al. (2013) kertovat artikkelissaan, että laadunvalvonnan toimenpiteiden suunnittelu on erittäin tärkeää hajautetussa ohjelmistonkehityksessä. Heidän mukaansa nimenomaan GSD:n näkökulmasta laadunvalvonnan johtaminen voidaan nähdä erilaisten käytäntöjen kokoelmana, jolla pyritään tuottamaan korkealaatuista ohjelmistoa. (Khan et. al. 2013).

Kumari et. al. (2014) mukaan ohjelmistojen *laadunvarmistus* (*Software Quality Assurance*, myöhemmin SQA) koskee koko ohjelmiston kehitysprosessia. SQA:n tarkoitus on estää ongelmien ja virheiden syntymistä valvomalla, että sovittuja käytäntöjä ja tapoja hyödynnetään ja löydetyt virheet korjataan. Ohjelmiston testaamisen tarkoitus on Kumari et. al. (2014) mukaan pääsääntöisesti yrittää rikkoa ohjelmistoa ja löytää sieltä virheitä, kun taas SQA keskittyy laadukkaiden ohjelmistoprosessien kehitykseen ja näin ollen estää virheiden syntymistä. Kumari et. al. (2014) esittävät artikkelissaan Carnegie-Mellonin Yliopistossa sijaitsevan Software Engineering Instituten kehittämän viisiportaisen organisaatioiden kypsyysmallin (*Capability Maturity Model,* myöhemmin CMM), jolla voidaan kuvata, millä tasolle yritysten ohjelmistonkehitysprosessit ovat. Tasolla yksi olevissa yrityksissä ohjelmistonkehitys on niin sanotussa kaoottisella tasolla. Tällä tasolla kaikkien sovittujen menetelmien puute johtaa siihen, että onnistunut projekti edellyttää projektiin osallistuvilta yksilöiltä lähes uskomattomia ponnistuksia. Tasot kolmesta viiteen tuottavat alempiin tasoihin verrattuna jo huomattavasti paremman laatuista ohjelmistoa. Tasolla viisi pystytään jo keskittymään jatkuvaan prossien parantamiseen ja ohjelmistonkehityksen käytännöt on kehitetty vastaamaan muuttuviin tarpeisiin. CMM ei kuitenkaan ota kantaa siihen, miten yritykset voivat parantaa omaa tasoaan. Sen tarkoitus on antaa yrityksille mahdollisuus ymmärtää ja määritellä omia prosessejaan. Kumari et. al. (2014).

3.1 Mitä laatu on

Näkökulmamme laatuun vaikuttaa siihen, miten laadun määrittelemme (Kitchenham &Pfleeger, 1996). Kitchenham ja Pfleeger viittaavat tutkimuksessaan viiteen eri Garvinin kehittämään näkökulmaan:

- Näkymätön laatu. Jotain joka voidaan tunnistaa, mutta ei määritellä.
- Käyttäjän näkökulma. Miten palvelu sopii tarkoitukseensa.
- Valmistajan näkökulma. Miten tuotteen valmistus on sujunut vaatimuksiin verrattuna.
- Tuotenäkökulma.
- Arvopohjainen laatu. Riippuu siitä, miten paljon asiakas on valmis maksamaan tuotteesta.

Valittu näkökulma vaikuttaa myös siihen, miten laatua halutaan mitata. Valmistajan kannalta voidaan erotella kaksi ominaisuutta: virheiden määrä ja niiden korjaamisesta aiheutuvat kustannukset. Käyttäjän näkökulmasta tärkeimpiä ovat käytettävyys ja luotettavuus. On siis ymmärrettävä, että laatu on hyvin kontekstiriippuvaista, koska se tarkoittaa eri asioita eri sidosryhmille. Parantaakseen laatua pitää määritellä ne näkökulmat joista on kiinnostunut, ja päättää, miten niitä mitataan. (Kitchenham &Pfleeger, 1996).

3.2 Laadunvalvonta hajautetussa ohjelmistonkehityksessä

GSD -projektin tehokas koordinointi sen koko elinkaaren ajan, on vaativaa ja monimutkaista. Ohjelmistojen laadunvarmistus (SQA), tarjoaa käytäntöjä, joilla ohjelmistotuotteiden laatua voidaan parantaa. Ivček ja Galinac (2008) esittävät tutkimuksessaan parhaat SQA -käytännöt hajautetussa ohjelmistonkehityksessä. SQA:n tarkoitus on tarjota käytäntöjen kautta johdolle ymmärrys niistä prosesseista, joita ohjelmistoprojektissa käytetään. Parhaisiin käytäntöihin kuuluu muun muassa SQA –ryhmän kasaaminen. Ryhmä on vastuussa suunnitelmista ja toimintatavoista, jotka sekä tuovat lisäarvoa että asettavat rajoitteet projektille. (Ivček & Galinac, 2008).

Ivček ja Galinak (2008) kuvaavat tutkimuksessaan kahden yksikön välistä onnistunutta GSD –projektia. Projektin elinkaari on jaettu analyysi-, suunnittelu-, toteutus- ja loppuvaiheeseen. Analyysi- ja suunnitteluvaiheissa tunnistetaan projektiin kuuluvat sidosryhmät, analysoidaan projektin mahdollisuuksia, budjetoidaan, tehdään aikatauluarvio, määritellään projektin tavoitteet ja arvioidaan riskit. Näissä vaiheissa määritellään myös laatusysteemi, jotta projektin lopputulos saadaan vastaamaan sekä asiakkaan että organisaation laatutavoitteita. Asiakkaan vaatimuksista ja projektin toteuttamisesta tehdään kummastakin omat dokumenttinsa seuraavia vaiheita varten. Suunnitteluvaiheessa SQA –toiminnan kannalta määritellään roolit ja vastuut, sekä selkeät kommunikaatiopolut sidosryhmien välillä. (Ivček & Galinac, 2008).

Projektin toteutusvaiheessa projektin johto valmentaa, ohjaa ja kontrolloi työntekijöitä haluttuun suuntaan ja vastaa samalla muutosten hallinnasta. Laadunvalvonnan kannalta toteutusvaihe on monimutkaisin ja vaativin. Ivček ja Galinakin (2008) esimerkkitapauksessa laadunvalvonta hoidettiin käyttämällä seitsemää erilaista menetelmää.

- Laadunhallintaverkosto: Verkosto, joka koordinoi ja hallitsee laadunhallinnan aktiviteetteja muun muassa tapaamisin ja puhelimitse. (Ivček & Galinac, 2008).
- Muutosten hallinta: Muutosten hallintaa varten perustettu tiimi, joka ennalta määrättyjen kriteerien perusteella päättää mahdollisista muutoksista. (Ivček & Galinac, 2008).
- Aktiivinen riskienhallinta: Viikoittainen raportti mahdollisista riskeistä projektipäällikölle. (Ivček & Galinac, 2008).
- Laadun auditoinnit: Sisäiset tarkastukset tärkeimmille projektin osille. (Ivček & Galinac, 2008).
- Tarkastusstrategia: Painopiste tarkastuksissa laitettiin aikaisille manuaalisille tarkastuksille. Pyrittiin löytämään mahdolliset viat hyvin varhaisessa vaiheessa. (Ivček & Galinac, 2008).
- Toimitusstrategia: Tuotteen toimittamisen yhteydessä sovittu tapa hoitaa löydetyt virheet ja niiden korjaaminen. (Ivček & Galinac, 2008).
- Raportointi ja mittaaminen: Laadun tason seuraaminen ja sen ilmoittaminen kuukausittain projektipäällikölle. (Ivček & Galinac, 2008).

Viimeisessä, projektin lopetusvaiheessa, kootaan loppuraportti sekä siirretään projektin aikana opitut tiedot organisaatioon ja projekti suljetaan. (Ivček & Galinac, 2008).

Ivček ja Galinacin (2008) esimerkkitapauksen perusteella tehokkain tapa vähentää kustannuksia ja tukea projektin laadunhallintaa, on löytää puutteet mahdollisimman aikaisin. Heidän mukaansa laadunvalvontaprosessit – ja strategia ovat erittäin keskeisessä asemassa GSD –projekteissa. Onnistunut laadunvalvonta ja laatuvaatimuksiin pääseminen vaatii selkeää kommunikaatiota, SQA –toimintojen hyvää koordinointia sekä jatkuvaa mittausta ja seurantaa (Ivček & Galinac, 2008).

Khanin et. al. (2013) mukaan laadunvalvonnan toimenpiteiden suunnittelu on erittäin tärkeää globaalissa ohjelmistonkehityksessä. Heidän mukaansa laadunvalvontaan liittyviä suorituksia tehdään projektin analyysi-, suunnittelu- ja loppuvaiheessa. Analyysivaiheessa pääpaino on budjetoinnissa ja tehtävien jaossa tiimeille. Toteutusvaiheessa käytetään erilaisia laadunvalvonnan strategioita, kuten esimerkiksi laatuauditointeja, muutosten hallintaa, aktiivista riskienhallintaa ja suorituskyvyn mittauksia. Loppuvaiheessa tehdään yhteenveto projektin aikana saaduista kokemuksista. Budjetointivaiheessa voidaan esimerkiksi kiinnittää huomiota kustannuksiin, jolloin yksi painopisteistä on hankkia pätevää työvoimaa sieltä, missä se on halvinta. Toteutusvaiheessa voidaan mahdollisesti käyttää automatisoitua koodintarkastusta ja käyttää koodausstandardeja, jotta laatua saataisiin parannettua. Kuitenkin pitää muistaa, että tuotteen laadun arviointi hajautetussa ohjelmistonkehityksessä on erittäin hankalaa ja vaatii vielä lisätutkimuksia. (Khan et. al. 2013).

3.3 Puutteet ja niiden havaitseminen

Ivček ja Galinacin (2008) edellä mainittua teoriaa puutteiden aikaisesta löytämisestä tukee osaltaan Wagnerin (2006) artikkeli, jonka mukaan virheiden korjaaminen yksikkötestausvaiheessa on hyvin paljon edullisempaa kuin järjestelmätestauksen yhteydessä. Wagnerin löydöt viittaavat siihen, että yksikkötestaus olisi hyvin kustannustehokasta. Myös Khanin et. al. (2003) mukaan varhainen puutteiden havaitseminen vähentää projektin toteutusaikaa ja kustannuksia sekä parantaa tuotteen laatua.

Yhtenäistä mallia puutteiden luokitteluun ollaan yritetty tehdä niin akateemisessa kuin yritysmaailmassakin. Malleista löytyy samankaltaisuuksia, mutta yhtään niistä ei olla hyväksytty perustyökaluksi ohjelmistoprojekteissa (Wagner 2008). Jopa kansainvälinen tekniikan alan järjestö IEEE (Institute of Electrical and Electronics Engineer) on kehittänyt standardin ohjelmistopoikkeuksien luokitteluun (IEEE 2010). Tässä aliluvussa tutustutaan hieman tarkemmin siihen, miten puutteet havaitaan ja luokitellaan.

3.3.1 Ortogonaalinen virheluokittelu

Ortogonaalista virheluokittelua (*Orthogonal Defect Classification*, myöhemmin ODC) voidaan käyttää sovelluskehityksessä jo aikaisessa vaiheessa suunnittelun yhteydessä ennaltaehkäisemään virheiden syntyä (Chillarege et. al 1992). Koska GSD -projekteissa puutteiden varhainen löytäminen on tärkeää (Ivček & Galinac, 2008, Khan et. al. 2013), voi ODC olla hajautetussa ohjelmistonkehityksessä apuna puutteiden havaitsemisessa.

Ortogonaalisen virheluokittelun lähtökohtana on liittää virheet ennalta määrättyihin luokkiin. Luokat taas osoittavat kehitysprosessin eri vaiheisiin, jolloin virheiden esiintymistä ja kehittymistä voidaan seurata koko ohjelmiston kehitysprosessin ajan. Jokaiseen virheeseen liittyy aina *virheen tyyppi (type*) ja *laukaisin (trigger*). Turhan koodaamisen välttämiseksi virhetyypit on suunniteltu niin, että suurin osa virheistä pystytään löytämään suunnitteluvaiheessa. (Nicholls et. al. 2011).

ODC -mallissa puutteen virheluokituksen tekee ohjelmistokehittäjä, joka myös vastaa ohjelmasta löytyneen virheen korjaamisesta. Kehittäjä määrittää johtuuko virhe jonkin puuttumisesta, vai siitä, että jotain on tehty väärin. Virhetyypit on pyritty tekemään mahdollisimman yksiselitteisiksi, jotta ohjelmistonkehittäjän on helppo jakaa virheet oikeisiin tyyppeihin. (Nicholls et. al. 2011).

Virheen laukaisijoilla mitataan testaamisen ja todentamisen edistymistä ohjelmistonkehityksen elinkaaren aikana. Virheiden esille tulo johtuu virheiden laukaisijoista. Laukaisin kertoo ne olosuhteet ja tekijät, jonka takia virhe tulee esiin ja se löydetään. (Sullivan & Chillarege, 1991). Nicholls et. al (2011) ovat jakaneet virheiden laukaisijat kolmeen luokkaan:

1. *Katselmointi- ja tarkastustestit*: Ongelmat tunnistetaan suunnitteludokumentaatiosta ja koodista. Tämän virheluokan laukaisijat tulevat esiin, kun projektiin osallistuvat ihmiset käyvät läpi projektin osia ajatuksen kanssa. Projektiin osallistuvien ihmisten osaamistaso vaikuttaa löydettävien virheiden laatuun. (Nicholls et. al. 2011).
2. *Yksikkö- ja funktiotestit*: Ongelmat löydetään, kun ohjelmiston koodia ajetaan. Testejä varten suunnitellaan ja kirjoitetaan testisuunnitelmat. (Nicholls et. al. 2011).
3. *Järjestelmätestit*: Ongelmat löytyvät, kun järjestelmää testataan mahdollisimman oikeissa lopullisissa olosuhteissa. Järjestelmätestauksessa pyritään löytämään virheet, jotka todennäköisesti esiintyvät loppukäyttäjällä. Testaus suoritetaan useimmiten, kun suuri osa järjestelmästä on valmis. (Nicholls et. al. 2011).

ODC:n on tarkoitus antaa käyttäjille tietoa siitä, miten virheet syntyvät ja miten ne voidaan korjata kehitysvaiheessa testausprosessien ja tarkastamisen avulla (Nicholls et. al. 2011). Vaikka ODC on virheluokitteluna hyvin kattava ja sen kustannukset ohjelmiston elinkaaren aikana ovat pienet, tulee alkukustannuksiin varautua (Chillarege et. al. 1992). Jotta ODC saadaan hyötykäyttöön, tarvitaan alussa investointeja koulutukseen, työkaluihin ja muutoksiin prosesseissa (Chillarege et. al. 1992).

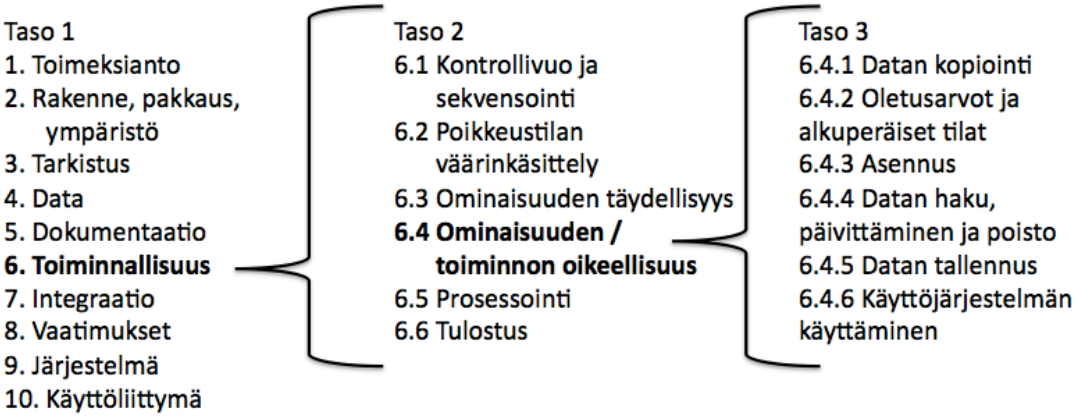
3.3.2 SAWO luokittelujärjestelmä

Itä-Suomen Yliopistossa kehitetty *SAWO* on toiminnallisten virheiden luokittelumenetelmä. Mallin kehitys juontaa juurensa aiemmin tehtyyn tutkimukseen, jota varten analysoitiin 11 879 ohjelmistopuutetta. Analyysin tarkoituksena oli selvittää, mitkä ovat yleisimmät puutetyypit, ja miten puutteet jakautuvat. Tutkimuksessa saatiin selville, että yleisimmät puutteet ovat toiminnallisia (65,5%), toisin sanoen puutteita laskenta- ja/tai toimintalogiikassa. Olemassa olevat virheluokittelumallit osoittautuivat liian yleisiksi, joten tutkijaryhmä alkoi niiden pohjalta rakentaa uutta mallia (Raninen et. al. 2012). Toiminnallisten ohjelmistopuutteiden luokitteluun keskittynyt alustava malli julkaistiin 2012 (Toroi et. al. 2012). Ensimmäinen virallinen SAWO niminen luokittelujärjestelmä julkaistiin 2013 (Toroi et. al. 2013). SAWO:n perim-

mäinen tarkoitus on kehittää ohjelmistoprosesseja käyttämällä ja analysoimalla puutetietoja (Toroi et. al. 2013).

SAWO mallissa puutteet luokitellaan kolmella eri tasolla. Ensimmäisellä tasolla käsitellään puutteita yleisesti, toinen taso keskittyy toiminnallisiin puutteisiin ja kolmas taso lisää vielä yksityiskohtia toiminnallisuuksiin. Mallia avulla järjestelmän vikoja voidaan tarkistella riittävällä tarkkuudella, mitä tarvitaan tuottamaan käytännöllisiä ja kohdennettuja ehdotuksia prosessien parantamiseen. (Toroi et. al. 2013).

Mallin ensimmäisellä tasolla puutteet jaetaan kymmeneen eri luokkaan. Toisella tasolla ensimmäisen tason vian tyyppi ”Toiminnallisuus” on jaettu kuuteen alatyyppiin. Näistä kuudesta alatyypistä ”Ominaisuuden / Toiminnon Oikeellisuus” tarkennetaan vielä kolmannella tasolla kuudella eri alatyypillä. Virheluokittelua SAWO mallin tyyliin on havainnollistettu kuvassa 1.



Kuva 1: SAWO virheluokittelumallin rakenne (Toroi et. al. 2013)

Virheluokittelumenetelmää on sovellettu käytännössä ja tulokset ovat olleet rohkaisevia. Luokittelun avulla on voitu tarjota yrityksille käytännöllisiä ja kohdennettuja parannusehdotuksia ohjelmistoprosessien kehittämiseen (Toroi et. al. 2012). SAWO virheluokittelumalli on vielä hyvin nuori, joten kokemuspohjaa laajasta käytöstä ei ole olemassa.

3.3.3 IEEE:n luokittelustandardi

IEEE:n standardi, *IEEE Standard Classification for Software Anomalies* (IEEE Std 1044-2009), tarjoaa yhdenmukaisen menetelmän ohjelmistojen poikkeuksien luokitteluun riippumatta siitä, mistä poikkeukset juontavat juurensa tai milloin ne kohdataan. Projektin, tuotteen tai systeemin elinkaareen vaiheella ei ole luokittelun kannalta merkitystä. Standardia voidaan hyödyntää erilaisten ohjelmistojen, esimerkiksi sulautettujen järjestelmien, käyttöjärjestelmien ja applikaatioiden, kehityksessä. Luokittelun tarjoaman tiedon käyttö ei rajoitu ainoastaan ohjelmistovirheiden syiden analyysiin, vaan sitä voidaan käyttää moneen muuhunkin tarkoitukseen, joista mainittakoon muun muassa projektinhallinta ja ohjelmistokehitysprosessien parantaminen. (IEEE 2010).

Standardin tarkoitus on tuottaa yhteistä sanastoa, jonka avulla voidaan kommunikoida tehokkaasti ohjelmistojen poikkeuksista. Tarkoitus on myös vakiinnuttaa yhteisiä ominaisuuksia virheidenanalysointimenetelmien tukemiseksi. (IEEE 2010). Standardiin liittyy olennaisena osana alla mainitut käsitteet:

Puute *(defect*): Epätäydellisyys tai vajaavaisuus tuotteessa, jolloin tuote ei vastaa sille asetettuja vaatimuksia tai määrittelyjä. Tuote pitää korjata tai korvata (IEEE 2010).

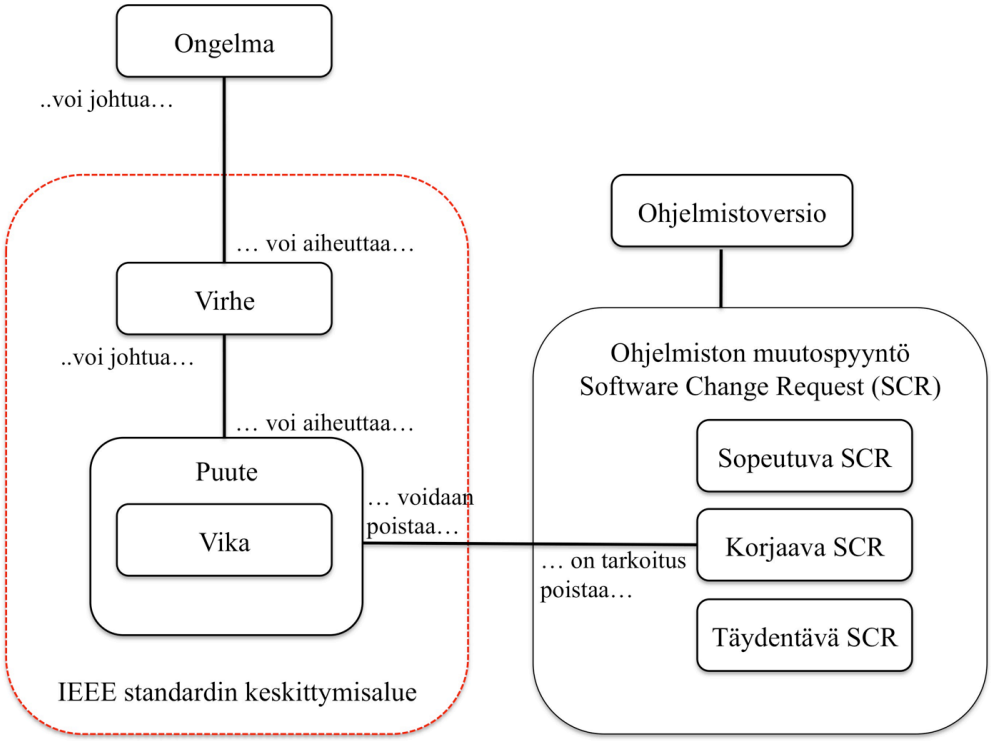
Erehdys *(error):* Inhimillinen toiminta, joka saa aikaan virheellisen lopputuloksen. Esimerkiksi väärin kirjoitettu koodi (IEEE 2010).

Virhe *(failure):* (A) Tuotteen kyky suorittaa haluttu toiminto kokonaan, tai tietyissä rajoissa, loppuu. (B) Tapahtuma, jolloin järjestelmä, tai sen osa, ei suorita haluttua toimintoa määrättyjen raamien puitteissa (IEEE 2010).

Vika *(fault):* Erehdyksen ilmentymä ohjelmistossa (IEEE 2010).

Ongelma *problem*: (A) Yhden tai useamman henkilön kokema hankaluus tai epävarmuus käyttäessä järjestelmää. (B) Hankala käyttötilanne, josta henkilön pitää suoriutua (IEEE 2010).

Kuvassa 2 olevat suorakulmiot kuvaavat standardin kokonaisuuksia (kiinnostuksen kohteita) ja viivat laatikoiden välissä kuvaavat kokonaisuuksien välisiä suhteita.



Kuva 2: Suhdekaavio IEEE Std 1044-2009 kokonaisuuksien välillä

Kuvasta 2 nähdään, että ongelmat voivat olla esiaste virheiden tunnistamiselle ja virhe voi johtua viasta. Itse ongelmien, ohjelmiston muutospyyntöjen ja ohjelmisto-versioiden luokittelu, ei kuulu IEEE 1044-2009 standardin alle. Nuo kokonaisuudet on otettu mukaan kuvaan selvyyden takia. IEEE:n standardi on kiinnostunut edellä mainittujen kokonaisuuksien suhteista, mutta keskittyy kuvassa 2 esitetyn punaiselle rajatun alueen sisään ja siellä olevien kokonaisuuksien luokitteluun. (IEEE 2010).

Standardin mukaan organisaation tulee luokitteluprosessia varten määrittää seuraavat asiat (IEEE 2010).

- Tavoitteet, jotka halutaan saavuttaa puutteiden ja virheiden luokittelulla (IEEE 2010).

- Vertailustandardi (esim. spesifikaatio, sopimus tai suunnitelma), jota käytetään kuvaamaan, mitkä systeemien tai ohjelmistojen ominaisuudet luetaan virheiksi (IEEE 2010).
- Miten erimielisyydet ja konfliktit luokittelua koskien ratkaistaan (IEEE 2010).
- Koska luokittelu aloitetaan ja lopetetaan projektin tai tuotteen elämänkaaren aikana (IEEE 2010).
- Projekti-, organisaatio- tai tuote –spesifit arvot, joita tarvitaan virheluokittelun ominaisuuksien määrittelyyn (IEEE 2010).
- Päätettävä kuka määrittää luokitteluominaisuuksien arvot löydetyille puutteille ja virheille (IEEE 2010).
- Missä ja miten luokitteludataa ylläpidetään (IEEE 2010).

Kun luokitteluprosessia varten määriteltävät asiat on sovittu, voidaan luokittelu aloittaa. Ohjelmistopoikkeuksien luokitteludataa voidaan käyttää myös laadun ominaisuuksien arvioinnissa. (IEEE 2010).

3.3.4 Ohjelmiston tarkastusprosessi GSD:ssä

Mishra & Mishra (2012) esittävät artikkelissaan hajautetussa ohjelmistonkehitysympäristössä tapahtuvaa tarkastusprosessia laadunvarmistuksen ja laadunhallinnan kannalta. Heidän näkemyksensä mukaan ohjelmistojen tarkastus on erittäin merkittävässä roolissa ohjelmistojen laadunvarmistusprosessissa. Ciolkowski et. al. (2003) mukaan laadunvalvonnan tehtävien, kuten esimerkiksi tarkastusten, arvioiden ja läpikäyntien, tarkoitus on löytää ohjelmiston puutteet tehokkaasti ja varhaisessa vaiheessa, jolloin tuotteiden laatua voidaan parantaa ja kehityskustannuksia vähentää. Ciolkowski et. al. (2003) painottavat, että yritysten tulee integroida tarkasteluprosessit osaksi kehittämisprosessia ja tarkasteluja tulee käydä läpi systemaattisesti. Tarkastelut vaativat myös jatkuvaa optimointia. Ciolkowski et. al. (2003) suosittelevatkin yrityksiä seuraamaan ja suorastaan matkimaan toisia yrityksiä, jotka ovat onnistuneet ottamaan tarkastukset ja läpikäynnit onnistuneesti käyttöön. Mishra & Mishra (2012) mukaan ohjelmiston tarkastuksen tarkoitus on löytää mahdolliset puutteet, ei korjata niitä. Heidän näkemyksensä mukaan tarkastuksesta saatuja tietoja voidaan käyttää laadunvarmistuksen seuraamiseen.

Caivano et. al. (2001) mukaan perinteisiä tarkastusprosesseja ei voi suoraan hyödyntää hajautetussa ohjelmistonkehityksessä. Perinteisemmät metodit pohjaavat hyvin vahvasti suoraan tiimin sisällä kasvokkain tapahtuvaan kanssakäymiseen, jota hajautetussa kehityksessä ei samalla mitalla ilmene. Tarkastusprosessit tulee Caivano et. al. (2001) artikkelin mukaan suunnitella uudelleen sellaisiksi, että ne tukevat kommunikaatiota internetin välityksellä. Myös Hedberg & Harjumaa (2002) ovat samoilla linjoilla. Heidänkin mielestään perinteiset ohjelmistojen tarkistusmetodit sopivat huonosti hajautettuun ohjelmistonkehitykseen. Yhtenäisen ajan ja paikan järjestäminen, joita tarvitaan usein, projektiin osallistuville on hajautetussa ohjelmistonkehityksessä hyvin hankalaa. Hedberg & Harjumaa (2002) tarjoavat tähän ratkaisuksi virtuaalista ohjelmiston tarkastusta. Virtuaalinen ohjelmiston tarkastus on prosessi, joka sopii projektin kulkuun ja tehdään hajautetusti käyttäen apuna tarkistustyökalua. Itse työkalu on tarkistusta varten suunniteltu ohjelmisto, jonka avulla voidaan vähintään hallita tarkastukseen liittyvää dokumentaatiota, ylläpitää tietoja puutteista ja kerätä automaattisesti dataa haluttujen mittarien avulla. Ja kaikki luonnollisesti verkon välityksellä hajautetusti. Hedberg & Harjumaa (2002) nostavat esiin kolme tärkeintä asiaa virtuaalisen tarkastuksen kannalta:

- Työkalut. Tarkastukseen suunniteltujen työkalujen tulee mahdollistaa prosessien tehokas toteuttaminen. Verkossa toimivien työkalujen avulla voidaan saavuttaa ajasta ja paikasta riippumaton tiedonhallinta. (Hedberg & Harjumaa 2002).
- Prosessien ja työkalujen joustavuudella varmistetaan riittävä käyttöönottotaso ja käytettävien menetelmien hyväksyminen organisaatiossa. (Hedberg & Harjumaa 2002).
- Tarvitaan prosessien ja työkalujen yhteentoimivuutta, jotta halutut menetelmät saadaan päivittäiseen käyttöön ja tarkastukset saadaan tehokkaiksi. (Hedberg & Harjumaa 2002).

Virtuaalinen ohjelmiston tarkastus sisältää useimmiten verkon kautta tapahtuvaa synkronista ja asynkronista kommunikaatiota, mutta myös kasvokkain olevia tapaamisia voidaan järjestää (Hedberg & Harjumaa 2002 ,Mishra & Mishra 2012).

Mishra & Mishra (2012) jakavat hajautetun ohjelmistontarkastusprosessin viiteen vaiheeseen:

- Valmistelu. Ensimmäisessä vaiheessa tarkastusryhmän johtaja valitsee ryhmän jäsenet ja tekee tarkastussuunnitelman. Tarkistettava dokumentti ja muut tarpeelliset asiakirjat tallennetaan tarkastustyökaluun. Tarkastustyökaluun voidaan asettaa vastuut, määräajat, tiedot suunnitellusta tarkastuksesta ynnä muuta tarvittavaa dataa. Tiedot voidaan toimittaa ryhmän jäsenille myös muulla tavalla. (Mishra & Mishra 2012).
- Yksittäisen tarkastusten vaihe. Tiimiin kuuluvat tarkastajat käyvät tuotteen läpi työkalussa olevan tarkistuslistan mukaisesti. He merkkaavat omat kommenttinsa työkaluun. Yksittäisen tiimin jäsenet eivät näe tässä vaiheessa toisten kommentteja. Ainoastaan tarkastusryhmän johtaja näkee kaikkien kommentit. (Mishra & Mishra 2012).
- Tapaamisvaihe. Kolmannessa vaiheessa koko ryhmä ja ryhmänjohtaja pitävät palaverin työkalun välityksellä. He keskustelevat löydetyistä puutteista. Keskustelun perusteella oikeat puutteet tunnistetaan ja väärät puutteet poistetaan vikaluettelosta. Ryhmänjohtaja koostaa lopullisen listan puutteista ja lähettää sen eteenpäin ohjelmiston tekijöille. (Mishra & Mishra 2012).
- Muokkausvaihe. Toiseksi viimeisessä vaiheessa tekijät käyvät listan läpi ja korjaavat puutteita. Jokaiseen puutteeseen lisätään selitys siitä, mitä on tehty ja missä paikassa. (Mishra & Mishra 2012).
- Seurantavaihe. Tässä vaiheessa tarkastusryhmän johtaja, tai joku tarkastajista, käy läpi muutokset ja varmistaa, että kaikki puutteet on korjattu. Mikäli jotain on vielä korjattavaa, palauttaa ryhmänjohtaja prosessin edelliseen vaiheeseen. (Mishra & Mishra 2012).

Koska hajautettu ohjelmistonkehitys lisääntyy, tulee virtuaalisen tarkastuksen tarve myös kasvamaan. Verkkoteknologioita hyödyntämällä on tarkastusten teko mahdollista myös hajautetuissa projekteissa. (Mishra & Mishra 2012). Stein et. al. (1997) mukaan hajautettu, asynkroninen ohjelmiston tarkastus on sekä toteutettavissa että kustannustehokasta.

4 GSD eri lähestymistapoja

Hajautettua ohjelmistonkehitystä voidaan tehdä niin ulkoiselle asiakkaalle, kuin yrityksen sisällä itselle. Tässä luvussa käydään läpi esimerkkitapaus ohjelmistonkehityksestä yrityksen sisällä, kun muualla tekstissä käsitellään lähinnä toiselle yritykselle ulkoistettua kehitystä Lisäksi luvussa perehdytään hieman enemmän ketterien menetelmien hyödyntämiseen hajautetussa ohjelmistonkehityksessä sekä niin sanottuun perinteiseen ohjelmistokehitykseen.

4.1 Ohjelmistonkehitys yrityksen sisällä

Bird et. al. (2009) tutkivat Windows Vistan kehitystä Microsoftin sisällä. Tutkimuksen tarkoitus oli selvittää ohjelmiston osien, joista osa tehtiin hajautetusti ja osa keskitetysti, laadullisia eroja. Lisäksi he tutkivat kehitysprosessia ja eri ilmiöitä kehityksen aikana. Tutkimuksessa keskitytään julkaisun jälkeen ilmenneisiin puutteisiin. Puutteiden määrittely tutkimuksessa vastaa IEEE standardin mukaista: ”Epätäydellisyys tai vajaavaisuus tuotteessa, jolloin tuote ei vastaa sille asetettuja vaatimuksia tai määrittelyjä (IEEE 2010)”. (Bird et. al. 2009).

Bird et. al. (2009) tutkimuksen tulokset olivat rohkaisevia hajautetun ohjelmistokehityksen kannalta. Heidän julkaisunsa mukaan yrityksen sisällä kirjoitetun ohjelmiston puutteiden määrässä ei ollut merkityksellistä eroa, oli koodi sitten kirjoitettu hajautettujen tiimien avulla tai keskitetysti yhdessä paikkaa. Laatu ei siis huomattavasti eronnut kehitystavasta riippuen. Bird et. al. esittävät muutamia huomioita, joista tämä voi johtua. (Bird et. al. 2009).

- Keskinäisen kilpailun puute. Kehityspaikat, joissa ohjelmistoa tehtiin, olivat olleet olemassa jo hyvän aikaa. Tekijät olivat työskennelleet saman ohjelmiston parissa vuosia. Ei ollut olemassa välitöntä uhkaa, että mikäli yksi kehityspaikka menestyy huonosti, se lakkautetaan. Tiimit pystyivät auttamaan toisiaan, eikä heidän tarvinnut todistella omaa erinomaisuuttaan. (Bird et. al. 2009).

- Suurin osa hajautetusta kehityksestä tapahtui Yhdysvalloissa ja Intiassa. Projektin alkuvaiheessa pitkään yrityksen palveluksessa olleet kehittäjät ja johtajat, jotka ymmärsivät hyvin yrityksen kehitysprosessit, lähtivät Intian kehityspaikalle. Lisäksi suuri osa heistä oli alun perin kotoisin Intiasta. Tällä tavalla organisatorisia ja paikalliskulttuurillisia muureja saatiin häivytettyä. (Bird et. al. 2009).
- Kommunikaation tärkeyttä nostettiin esiin. Työntekijät ottivat vastuulleen pienentääkseen aikaeron aiheuttamia haasteita. He saapuivat töihin aikaisin, tai olivat töissä myöhään, jotta pystyivät järjestämään telekonferensseja toisella mantereella olevien kanssa. Tärkeimpiin tapaamisiin järjestettiin säännöllisesti matkoja. (Bird et. al. 2009).
- Kehityksessä kaikki käyttivät johdonmukaisesti samoja työkaluja. Esimerkiksi: lähdekoodin hallintatyökalu, kehitysympäristö ja dokumentointimenetelmä. (Bird et. al. 2009).
- Organisaation yhtenäisyys. Maantieteellisesti eri paikoissa toimivilla tiimeillä on usein yksi johtaja. Eri lokaatioissa työskentelevät kehittäjät eivät joudu raportoimaan paikalliselle johtajalle. Tällä tavalla eri paikoissa olevat kehittäjät saadaan enemmän osaksi projektia ja yritystä. (Bird et. al. 2009).

Bird et. al. (2009) mukaan hajautetulla ohjelmistonkehityksellä ei siis välttämättä ole laadullista eroa verrattuna yhdessä maantieteellisessä paikassa tapahtuvaan kehitykseen (Bird et. al. 2009). Maantieteellisen eron sijaan suurempi merkitys laatuun vaikuttaisi olevan organisaation kypsyydellä ja mallilla (Bird et. al. 2009, Nagappan et. al. 2008).

4.2 Kehitys ketterillä menetelmillä

Kuten aiemmin on jo todettu, kommunikaatiolla on hyvin suuri merkitys GSD:n onnistumisen kannalta. Layman et. al. (2006) tutkivat ketterän ohjelmistokehityksen erään metodologian, *Extreme Programming* (myöhemmin XP), käyttöä hajautetussa ohjelmistonkehitysprojektissa, jossa onnistumisen edellytyksenä oli epäformaali ja rikas kommunikaatioympäristö. Projektin johto ja toteuttava tiimi olivat maantieteel-

lisesti kaukana toisistaan, XP oli ensimmäistä kertaa koko tiimin käytössä ja vaatimukset vaihtuivat useasti projektin aikana.

XP on iteratiivinen, ketterä kehitysmenetelmä, jossa suunnittelu- ja julkaisujaksot ovat melko lyhyitä. Yksi iteraatio kestää 1-2 viikkoa ja julkaisu pyritään saamaan valmiiksi 3 kuukauden sisällä. Julkaisu vastaa vakaata versiota tuotteesta, jota asiakas voi käyttää. Iteraatio on lyhyempi askel julkaisun kehityksessä, jossa kehittäjille annetaan tietty tehtävä hoidettavaksi. Ennen kehityksen aloittamista ei XP:ssä tarvita kattavia vaatimusmäärittelyjä tai muita suunnitteludokumentaatioita. Iteraatioita tarkastellaan ja arvioidaan projektiin osallistuvien sidosryhmien kesken, ja XP vaatiikin hyvin paljon jatkuvaa kommunikaatiota ja palautetta sidosryhmiltä, jotta muutoksiin voidaan sopeutua. Varsinkin asiakkaan rooli on erittäin tärkeä palautteen ja päämäärän antajana. (Layman et. al. 2006).

Layman et. al. (2006) esittävät artikkelissaan omat arvionsa ja suosituksensa siitä, miten luoda ympäristö, jossa hajautettu ohjelmistonkehitys ketterillä menetelmillä saadaan toimivaksi.

1. Asiakas. Jotta päätöksiä voidaan tehdä tehokkaasti ja vaatimukset saadaan määriteltyä, on asiakkaan roolin määritys olennaista. Ideaalitilanteessa asiakas on jatkuvasti tavoitettavissa ja fyysisesti paikalla vastatakseen kehittäjien kysymyksiin ja tehdäkseen päätöksiä. Hajautetussa ohjelmistonkehityksessä asiakkaan aktiivinen osallistuminen on erittäin kriittistä projektin onnistumisen kannalta. Layman et. al. (2006) suosittelevat, että päätös siitä, kuka on asiakas, tulee tehdä ennen projektin aloittamista. Valitulla henkilöllä on oltava mahdollisuudet tehdä päätöksiä hankkeen toiminnallisuuksista, hänen on oltava helposti tavoitettavissa ja hänellä on oltava omat intressinsä projektissa. (Layman et. al. 2006).
2. Yhdyshenkilö. Maantieteellisesti eri paikoissa olevissa GSD –tiimeistä, kannattaa valita avainhenkilö, joka edesauttaa kaksisuuntaista viestintää tiimien välillä. Kun esimerkiksi projektin johto ja kehitystiimi sijaitsevat eri paikoissa, kannattaa luoda ja antaa sopivalle henkilölle rooli, jonka tarkoitus on työskennellä läheisesti molempien tiimien kanssa. Yhdyshenkilöksi kannattaa

valita mahdollisuuksien mukaan sellainen, joka puhuu molempien tiimien äidinkieltä sujuvasti sekä ymmärtää kulttuurillisia eroja. (Layman et. al. 2006).

3. Asynkroninen kommunikaatio. Layman et. al. (2006) esimerkkitapauksessa projektin johto ja kehittäjät olivat kuuden aikavyöhykkeen päässä toisistaan. Aikaeron takia heillä oli normaalin työpäivän kuluessa vain kaksi tuntia ”yhteistä” aikaa kommunikoida suoraan toistensa kanssa. Koska ketterissä menetelmissä kommunikaatio asiakkaan ja kehittäjien välillä on hyvin tärkeässä roolissa, piti tiimien keksiä sopiva tapa kommunikoida keskenään asynkronisesti. Tähän ongelmaan löytyikin varsin nopeasti yksinkertainen ratkaisu. Tiimit käyttivät keskusteluissaan sähköpostituslistoja. Listojen avulla kehittäjät ja johto pysyivät ajan tasalla projektin etenemisestä ja ongelmista. Erityistä huomiota kiinnitettiin siihen, että kehittäjien esittämiin kysymyksiin vastattiin heti mahdollisuuden tullen. Asiakkaan nopeat vastaukset edesauttoivat kahta tärkeää asiaa. Kehittäjille osoitettiin, että asiakas on kiinnostunut projektista ja auttaa projektin etenemistä päivittäin. Toiseksi; asiakkaan ja kehittäjien välinen kommunikaatio saatiin päivittäiseksi rutiiniksi. Layman et. al. (2006) ehdottavatkin, että mikäli kommunikointi kasvotusten ei ole mahdollista, yksinkertaista sähköpostilistaa käyttämällä voidaan kommunikaatiota parantaa. Tärkeään rooliin listojen käytössä nousee mahdollisimman nopeat vastaukset kysymysten esittäjille. (Layman et. al. 2006).
4. Projektin hallintatyökalu. Projektin hallinta ja kontrollointi helpottuu, kun projektiin osallistuvat toimijat käyttävät yhteistä hallintatyökalua. Työkalun avulla parannetaan projektin läpinäkyvyyttä ja asiakkaalla on mahdollisuus seurata yksinäisten ominaisuuksien kehitystä. Tämä auttaa projektin ominaisuuksien priorisoinnissa ja aikatauluttamisessa. On huomioitava, että työkalu tarjoaa sidosryhmille juuri niin tarkkaa tietoa kuin käyttäjät sinne laittavat. On siis tärkeää pitää informaatio tarkkana ja ajantasaisena. (Layman et. al. 2006).

Riippumatta maantieteellisistä, ajallisista, kielellisistä ja teknisistä haasteista, eri menetelmillä on mahdollista luoda riittävän kommunikaatiorikas ympäristö, jotta ketteristä menetelmistä ainakin XP:tä voidaan käyttää hajautetussa ohjelmistonkehityksessä tehokkaasti. Käyttämällä edellä mainittuja metodeja ja suosituksia, kommunikaatio asiakkaan ja kehittäjien välillä on mahdollista saada sellaiselle tasolle, että

muutoksiin ja korjauksiin on mahdollista reagoida nopeasti ja tehokkaasti. (Layman et. al. 2006).

Useisiin ohjelmistoprojekteihin liittyy epävarmuustekijöitä niin vaatimusten kuin käytettävien teknologioiden osalta. Tällaisissa projekteissa selkeiden vaatimusmäärittelyiden saaminen on hankalaa, jolloin tarvitaan tiivistä yhteystyötä kaikkien asianosaisten kesken projektin aikana. Paasivaara & Lassenius (2006) mukaan ketterät menetelmät voivat olla ratkaisu tällaisissa hajautetuissa ohjelmistoprojekteissa.

4.3 Perinteinen ohjelmistokehitys

Cusumanon (2008) mukaan iteratiivisten tekniikoiden käyttö on hajautetussa ohjelmistonkehityksessä hankalaa, koska projektiin osallistuvat henkilöt ovat jakautuneet maantieteellisesti ja kulttuurillisesti. Iteratiivisessa kehityksessä muutoksiin pitäisi reagoida nopeasti, jolloin viestien pitäisi kulkea sujuvasti kehittäjien ja asiakkaan välillä. Tämä aiheuttaa organisatorisia ja teknisiä ongelmia iteratiiviselle kehitykselle. Siitä johtuen monet yritykset ovatkin käyttäneet hajautetussa kehityksessä perinteistä vesiputousmallia suunnittelun ja projektinjohdon apuna. Perinteisellä malli on nähty yksinkertaisena tapana helpottaa kommunikoinnin ja koordinoinnin ongelmia. Cusumano (2008) ehdottaa, että molempia malleja voitaisiin käyttää hyväksi GSD:ssä. Hänen mukaansa paras tapa on aloittaa kehitys iteratiivisesti, käyttää perinteistä vesiputousmallia toteutusvaiheessa ja palata taas iteratiiviseen malliin projektin loppupuolella. Tällainen kehitystapa vaatii ohjelmiston arkkitehtuurilta paljon. Jotta näin voidaan menetellä, pitää järjestelmä hajottaa selkeisiin osajärjestelmiin ja moduuleihin. Moduulien pitää olla sellaisia, että erikseen työskentelevät tiimit voivat ainakin jossain määrin suunnitella, rakentaa ja testata näitä moduuleita. Kehitystapa vaikuttaa osittain iteratiiviselta ja puhutaan niin sanotuista minivesiputousmallista. (Cusumano 2008).

Herbsleb & Grinter (1999) tekemän tutkimuksen mukaan on mahdollista, että hajautetussa kehityksessä tarkat suunnitelmat, prosessit ja määritelmät ovat hyödyllisiä. Kun on kyse suurista ohjelmistoprojekteista, ongelmat ovat heidän mukaansa hyvin samankaltaisia sekä hajautetussa että tietyssä paikassa tehdyssä ohjelmistonkehityksessä. Vaikka suunnitelmat, prosessit ja määritelmät olisi hyvin dokumentoitu, muu-

toksia kehityksen aikana tulee tapahtumaan. Vaikka kehitysprosessi ei olisi suoraan iteratiivinen, nämä muutokset pitää selkeästi kommunikoida kaikkien projektiin osallistuneiden kesken. Voidaan todeta, että kommunikaation järjestäminen hajautetussa kehityksessä on hankalampaa, mutta ei mahdotonta. Tärkeintä on valita kehitykseen sopiva prosessi. (Herbsleb & Grinter 1999).

5 KULTTUURI, KOMMUNIKOINTI JA ORGANISAATIO

Kompleksisissa kulttuurisissa järjestelmissä kansallisen kulttuurin ominaisuudet ovat tärkeitä, mutta eivät ainoita vaikuttajia organisaatiokulttuurin muodostumisessa. Tämä tarkoittaa, että organisaation kulttuuria voidaan hajautetussa ohjelmistonkehityksessä luoda ja muokata. Kulttuurin luominen ei kuitenkaan ole helppo tehtävä ja useita kulttuureita yhdistävissä projekteissa lopputulosta on hyvin hankala ennustaa. Kulttuurien yhteensovittamisen prosessia voidaan kuitenkin ymmärtää, mikä tarjoaa työkaluja edistää kommunikaatiota hajautettujen tiimien kesken. (Brannen & Salk, 2000).

Yksinkertainen esimerkki kommunikaatiotapojen erosta käy selville Herbsleb & Grinterin (1999) artikkelista. Hajautetussa kehityksessä englantia äidinkielenään puhuvat käyttivät kommunikaatioon mielellään puhelinta, koska he pystyivät sen kautta esittämään kysymyksiään ja tarkentamaan niitä samalla. Ne, joilla äidinkielenä oli jokin muu kuin englanti, eivät taas käyttäneet puhelinta mielellään. Heidän mielestään asiaa oli vaikea selittää tarkasti vieraalla kielellä puhelimessa. He suosivat mieluummin sähköpostia, koska kirjoittamiseen pystyi käyttämään aikaa, jolloin oman asiansa sai tuotua selkeästi esille. (Herbsleb & Grinter 1999).

Sangwanin &Rosin (2008) mukaan hajautetussa ohjelmistonkehityksessä osa kommunikaatio-ongelmista on kulttuurijohdannaisia. He nostavat projektiarkkitehdin roolin arvokkaaksi erilaisten kuilujen yhdistäjänä.

Kuten useasti on jo todettu, yksi tärkeimmistä asioista onnistuneeseen hajautettuun ohjelmistonkehitykseen on sujuvassa kommunikaatiossa ja projektinhallinnassa. Tässä luvussa käydään läpi kulttuurillisia, henkilösuhteisiin ja organisaatioon liittyviä seikkoja, joiden avulla kommunikaatiota ja näin ollen ohjelmiston laatua voidaan parantaa.

5.1 Kulttuurien johtaminen

Krishna et. al. (2004) ovat tutkineet kulttuurien välisistä eroista johtuvia haasteita kansainvälisesti hajautetussa ohjelmistonkehityksessä. Tutkimuksen empiirinen pääpaino on ollut Länsi-Euroopan, Pohjois-Amerikan ja Japanin ulkoistama sovelluskehitys Intiaan. Heidän ensisijainen huomionsa on ollut, että ulkoistamisen yhteydessä kulttuurien väliset erot aiheuttavat ongelmia. Vaikeudet johtuvat muun muassa yhteisöjen erilaisista tavoista työskennellä ja niin sanottujen sillanpäätiimien työskentelystä toisessa kulttuuriympäristössä. Työskentelytapojen lisäksi haasteita asettavat sosiaalisen käytöksen kulttuuriset normit, vieras kieli ja suhtautuminen auktoriteetteihin. (Krishna et. al. 2004).

5.1.1 Projektien valinta

Yksi lähestymistapa kulttuurienvälisten projektien hallintaan on kiinnittää huomiota siihen, millaisia projekteja valitaan kehitettäväksi hajautetussa ympäristössä. Krishna et. al. (2004) mukaan esimerkiksi kuluttajatuotteisiin suunnatut sulautetut järjestelmät voidaan usein määritellä kulttuurillisesti hyvin neutraalilla tavalla, jolloin kulttuurienvälistä ymmärrystä ei tarvita kovinkaan paljon. Projektien valinnassa kannattaa ottaa huomioon myös se siihen, miten arvokasta projektista saatava oppi on. Mikäli tietyn alan asiantuntemusta on saatavilla ja tietotaitoa saadaan siirrettyä yritykseen, projekti saattaa hyvinkin olla kannattava. (Krishna et. al. 2004).

5.1.2 Suhteiden hallinta

Asiakkaan ja toimittajan välisen suhteen hallinta ja aktiivinen johtaminen on ensisijaisen tärkeää suhteen molemmin puolin (Krishna et. al. 2004). Suhteen hallintaa voidaan helpottaa muun muassa käyttämällä yhteisiä järjestelmiä (Heeks et. al. 2001, Gopal et. al. 2002, Layman et. al. 2006). Sovittujen koordinointi- ja kontrollointimekanismien lisäksi (Krishna et. al. 2004), suhteiden hallintaa voidaan parantaa yhteisillä kehitysmetodeilla ja välttämällä keskenään riitelevien teknologioiden käyttöä (Gopal et. al. 2002). On kuitenkin ymmärrettävä, että arvot ja normit pohjautuvat erilaisiin kulttuurillisiin taustoihin, työelämään ja koulutukseen, joten niiden yhteensovittaminen ei tapahdu ongelmitta. Monikulttuurillisissa tiimeissä lähentymistä voidaan kuitenkin erilaisin keinoin edistää, jolloin tiimien jäsenet ottavat toisen kulttuu-

rin toimintatapoja itselleen. Tällainen on aikaa vievä prosessi, jossa voidaan käyttää esimerkiksi edellä mainittuja sillanpäätiimejä. Tiimit viettävät huomattavan osan ajasta asiakkaan tiloissa. Valitulla henkilöstöllä ja heidän kouluttamisellaan on iso rooli kulttuurien yhdistämisessä. (Krishna et. al. 2004)

5.1.3 Henkilöstö ja koulutus

Eritoten ohjelmistojen kehityksessä syvällinen asiakkaan ja toimittajan välinen yhteistyö on tärkeää. Jotta yhteistyö olisi hedelmällistä, tulee henkilöstön valintaan käyttää aikaa. Onnistuneita ulkoistamisprojekteja johtamaan on usein valittu eri kulttuureita yhdistävä ihminen. Tällainen voi olla esimerkiksi Yhdysvalloissa kouluttautunut ihminen, joka on kotoisin toisesta maasta. Koska hän tuntee molempien kulttuurien tapoja, pystyy hän paremmin myös ymmärtämään niiden eroista syntyviä ongelmia. Ohjelmiston toimittaja voi mahdollisuuksien mukaan myös lähettää kehittäjiä asiakkaan toimitiloihin ja luoda näin monikulttuurisen tiimin, jossa on sekä paikallisia että ulkomaisia työntekijöitä. (Krishna et. al. 2004).

Monikulttuurillisissa asiakas-toimittaja suhteissa kulttuurien koulutus on usein nähty tarpeelliseksi vain toimittajalla. Krishna et. el. (2004) mukaan tämä on kuitenkin väärä tapa. Opetuksen tulisi tapahtua molempiin suuntiin, jotta kaikki kulttuurilliset aspektit voidaan ottaa huomioon. Koulutus usein myös lopetetaan liian aikaisin. Opetus aloitetaan ennen projektia ja saadaan valmiiksi projektin alkaessa. Myös projektin aikana tulisi olla koulutusta, joka tukee muun muassa kokemusten ja oppien jakoa työtovereiden kesken. Kun asiakkaan ja toimittajan kulttuurillinen koulutus aloitetaan ennen projektia ja sitä jatketaan projektin ajan, voidaan arvoista ja työtavoista johtuvia eroja johtaa koko tuotteen elinkaaren ajan. (Krishna et. al. 2004).

5.2 Organisaatio

Niin sanotun Conwayn lain mukaan, joka on saanut nimensä Melvin Conwaylta, organisaatiot tuottavat sellaisia järjestelmiä, jotka kuvastavat niiden kommunikaatiorakenteita (Nagappan et. al. 2008, Herbsleb & Grinter 1999). Tämä tarkoittaa, että organisaation kommunikaatiotavat jättävät lähtemättömän jäljen valmistettuun tuotteeseen (Herbsleb & Grinter 1999). Nagappan et. al. (2008) tutkimus tarjoaakin empii-

ristä näyttöä siitä, että organisaation eri mittarit voivat tehokkaasti indikoida vikaalttiutta. Mitä aiemmin ohjelmiston kehittäjät saava tiedon mahdollisista ongelma-alueista, sitä helpompi niihin on puuttua. Nagappan et. al. (2008) mukaan ongelmakohtien arvioinneissa yksi tärkeimmistä tekijöistä ovat ihmiset ja organisaatio. Organisatorisista mittareista voidaan seurata esimerkiksi: tietyn moduulin kehittäjien määrää, kuinka paljon kehittäjät joutuvat tekemään samanaikaisesti eri tehtäviä, kuinka paljon moduuliin tulee muutoksia organisaation sisältä ja kuinka etäisiä kehittäjät ovat toisilleen organisatorisesti. Näitä mittareita seuraamalla voidaan kehitettävästä ohjelmistosta tehdä vika-arvio aikaisessa vaiheessa, jolloin päätökset testaamisesta, uusista suunnitelmista, koodin tarkastuksista ja testauksesta voidaan myös tehdä mahdollisimman aikaisin. (Nagappan et. al. 2008).

Sangwan & Rosin (2008) mukaan GSD -organisaatiossa kaikkien sidosryhmien esille tuomat asiat tulee yhdenmukaisesti ja harmonisoidusti kommunikoida kaikille projektiin osallistuville tiimeille. Mikäli kehityksen halutaan olevan tehokasta, pitää eri työyksikköjen riippuvuudet toisiinsa olla selkeästi ymmärrettyjä. Työtehtävien jako hajautetuille tiimeille pitää olla huolellisesti suunniteltu. Sangwan & Ros (2008) näkemyksen mukaan projektin arkkitehti on organisaatiossa tärkeässä asemassa. Hänen tulee kommunikoida projektin tavoite, ongelma joka ratkaistaan, tehokkaasti kaikille projektiin osallistuville. Hänen tehtävänsä on myös käyttää koko organisaation yhteistä, jaettua tietoa, tehdäkseen oikeita ja tehokkaita ratkaisuja projektin edistämiseksi. (Sangwan & Ros 2008).

Brannen & Salkin (2000) tutkimuksen mukaan monikansallisissa organisaatioissa on mahdollista saavuttaa yhtenäinen yrityskulttuuri. He näkevät organisaation kulttuurin dynaamisena jatkuvana prosessina, jossa käydään sosiaalisia neuvotteluita. Syntyvä kulttuuri on näiden neuvotteluiden tulos. Yksilön kulttuurilliset kannanotot tulevat esille heidän neuvottelukäyttäytymisessään. Kulttuurilliset kannanotot tulee suhteuttaa yksilön asemaan ja vaikutusvaltaan meneillä olevassa projektissa. Näin voidaan saada jonkinlainen kuva siitä, miten yksilö voi vaikuttaa organisaation yhtenäisen yrityskulttuuriin muodostumiseen. (Brannen & Salk 2000).

6 JOHTAMINEN

GSD:n johtaminen asettaa useita eri haasteita muun muassa organisaatiolle, muutosten hallinnalle ja projektinhallinnalle. Jopa riski, että tuottavuus laskee, on olemassa. GSD tarjoaa silti paljon hienoja mahdollisuuksia liittyen esimerkiksi innovaatioihin ja prosessien parantamiseen. (Ebert & Neve, 2001). Beulenin ja Ribbersin (2002) mukaan IT:n ulkoistaminen itsessään on hyvin monimutkaista ja vaatii paljon johdolta. Heidän artikkelinsa mukaan organisaatioiden välisen kumppanuuden johtaminen on pääasiassa hallinnollinen ongelma.

Tässä luvussa tehdään katsaus GSD:n aiheuttamiin hallinnollisiin ja johtamisen ongelmiin.

6.1 Tiedonhallinta

Desouza et. al. (2006) mukaan globaalissa ohjelmistonkehityksessä tavallisia tiedonhallinnan ongelmia ovat relevantin tiedon löytäminen, tiedon jakamisen ongelmat ja tiedon yhdistämisen vaikeudet. Tiedon hallintaa pitää johtaa koko ohjelmistonkehityksen elinkaaren ajan. Tiedon johtamisella saadaan myös parannettua yrityksen ohjelmistokehityksen prosesseja ja käytäntöjä. (Desouza et. al. 2006).

Jotta tiedonhallinta onnistuisi GSD:ssä, pitää yrityksellä olla toimiva globaali strategia tiedon johtamiseen. Yrityksen valitseman strategian tulee johtua kolmesta eri tekijästä: millainen on tehtävän työn luonne, kuinka itsenäisiä maantieteellisesti eri paikoissa olevat toimistot/tiimit ovat ja kolmanneksi, millaisia resursseja eri paikoilla on käytettävissään tiedonhallinnan johtamiseen. Desouza et. al. (2006) esittelevät artikkelissaan kolme eri strategiaa tiedonhallinnan johtamiseen.

- *Keskitetty strategia*. Tiedonhallinnan päätökset tehdään yhdessä paikassa, tarjotaan teknistä tukea ja hallinnoidaan ohjelman täytäntöönpanoa. Tämä strategia kannattaa valita, mikäli standardoidun työn määrä on suuri, hajautetut tiimit eivät ole itseohjautuvia ja organisatoriset resurssitarpeet ovat suuria. (Desouza et. al. 2006).

- *Osittain paikallinen strategia*. Yrityksen johto antaa laveat suuntaviivat ja periaatteet. Tiedonhallinnan toimeenpano ja yksityiskohdat jätetään alueellisten toimijoiden vastuulle. Strategia on suositeltava mikäli standardoidun työn ja itseohjautuvuuden määrä sekä organisatoriset resurssitarpeet ovat keskivertoa. (Desouza et. al. 2006).
- *Hajautettu paikallinen strategia*. Jokainen yksikkö sovittaa omat tiedonhallinnan toimensa tarpeisiinsa sopiviksi. Paikallinen tiedonhallinnan ohjaus pitää yllä yksiköiden välistä kommunikaatiota ja antaa periaatteita ja apua tiedonhallinnan johtamiseen. Tällainen strategia on sopivin, jos standardoidun työn määrä on pientä, paikalliset toimijat ovat hyvin itsenäisiä ja organisatoriset resurssien tarpeet ovat pieniä. (Desouza et. al. 2006).

Strategian valinnan jälkeen siirrytään *tiedonhallintajärjestelmän* (*Knowledge Management System*, myöhemmin KMS) käyttöönottoon. Desouza et. al. (2006) esittävät artikkelissaan kolme käytössä olevaa arkkitehtuurimallia tiedonhallintaan: *asiakas-palvelinmalli* (*client-server model*), *vertaismalli* (*peer-to-peer*), ja *hybridimalli* (*hybrid model*).

Asiakas-palvelinmallissa yritys yksinkertaisesti tallentaa informaatiota tietokantoihin, joista kuka tahansa yrityksen työntekijä pääsee niitä katsomaan. Käyttäjiä palkitaan tiedon jakamisesta ja hyödyntämisestä. Tässä mallissa IT –investoinnit ovat usein isoja. (Desouza et. al. 2006).

Vertaismallissa investoinnit informaatioteknologiaan ovat maltillisempia. Mikäli yhtiön kilpailukyky edellyttää luovia ratkaisuja strategisiin ongelmiin, saavutetaan tällaiset ratkaisut yhdistämällä ihmisiä ja heidän kokemustaan. Mallin avulla pyritään lisäämään keskustelua oikeiden ihmisten välillä. Henkilöstöä palkitaan osaamisen jakamisesta toistensa kesken. (Desouza et. al. 2006).

Hybridimalli, joka sopii GSD:n tiedonhallinnan tarpeisiin, yhdistää asiakas-palvelinmallin ja vertaismallin piirteitä. Tieto, jota voidaan hyödyntää koko organisaatiossa, tallennetaan keskitetysti. Tarpeellista on tallentaa data sellaisena, että sitä voidaan hyödyntää globaalisti poistamalla paikalliseen kehitykseen liittyvät asiayhteydet. Tällaista informaatiota ovat esimerkiksi projektien kustannus- ja hyötyanalyysit, tarvittava henkilöstömäärä ja sijoitetun pääoman tuotto. Jotta oikea, hyödylli-

nen tieto saadaan talteen, tulee projektiin osallistuvien tiedostaa mitä tehdään ja missä, miksi tehdään ja miten sekä mikä on kunkin projektiin osallistuvan rooli. Tiedon keskistetyllä tallentamisella ja hallinnalla on tarkoitus varmistaa jaetun tiedon ylläpito, parantaa tiedon etsimistä ja hyödyntämistä, edistää tiedon liikkumista projekteista muihin projekteihin, parantaa tiedon oikeellisuutta ja identifioida tiedon lähde. Hybridimallissa käytetään myös vertaismallista tuttua suoraa kommunikaatiota eri projektien välillä kehittäjien kesken. Keskitetysti tallennetusta datasta esimerkiksi ohjelmiston kehittäjä saa selville tiedon lähteen ja voi näin ollen olla suoraan yhteydessä häneen. Paikalliset toimijat voivat myös ylläpitää omia tietokantojaan, joista aika ajoin voidaan tutkia mikä olisi mahdollisesti tarpeellista myös muille ja lisätä tämän datan keskitetysti hallittuun kantaan. (Desouza et. al. 2006).

Varsinainen tiedonhallintajärjestelmän käyttöönotto ja valjastaminen hyötykäyttöön riippuu vahvasti organisaation kypsyydestä. Hyvin kypsissä globaalisti toimivissa ohjelmistoalan yrityksissä IT –osasto vastaa usein tiedonhallinnan infrastruktuurin rakentamisesta ja ylläpidosta. Tällaista mallia Desouza et. al. (2006) kutsuvat keskitetyksi malliksi. Niin sanotun jaetun palvelun mallissa teknologiaosasto tarjoaa tukea, apua ja henkilökuntaa KMS:n käyttöönottoa ja hyödyntämistä varten, mutta ei kontrolloi KMS:sää niin tarkasti kuin keskitetyssä mallissa. Kolmas Desouza et. al. (2006) esittämä malli on *tarkkailijamalli*. Tarkkailijamallissa teknologiaosasto ehdottaa ja rajoittaa tiettyjä tiedonhallinnan malleja eri yksiköille. Sekä jaetun palvelun malli että tarkkailijamalli ovat suositellumpia organisaatiolle, jotka eivät ole kovin kypsiä, tai organisaatioille, joiden sisällä tehtävässä ohjelmistonkehityksessä on suuria eroja. (Desouza et. al. 2006).

6.2 Hankkijan ja toimittajan välinen suhde

Beulen & Ribbers (2002) esittävät haastatteluihin ja kirjallisuuteen perustuvassa artikkelissaan yritysjohdon käytäntöjä monimutkaisissa IT:n ulkoistuksissa sekä toimittajan että hankkijan näkökulmasta. Jotta ulkoistus saadaan pitkäkestoiseksi onnistuvaksi prosessiksi, tarvitaan molempien osapuolten omistautumista asialle. Beulen & Ribbers (2002) mukaan organisaation välisten suhteiden hallinta on johdon hallinnollinen ongelma. Ongelmaa voidaan lähestyä muun muassa ymmärtämällä ja soveltamalla johtamisen yleisiä teorioita käytännössä opittuihin malleihin. Kompleksisten

toimittaja-hankkija suhteiden hallinnan tärkeimmiksi tekijöiksi Beulen & Ribbers (2002) nostavat: IT –strategian tiedonhallinnan, sopimukset ja sopimusten hallinnan sekä henkilöstöresurssien saatavuuden. Seuraavissa aliluvuissa tutustumme hieman tarkemmin edellä mainittuihin.

6.2.1 IT –strategia ja tiedonhallinta

Yrityksen IT –strategia kertoo, mikä on yrityksen pitkän tähtäimen visio siitä, miten IT:tä tullaan hyödyntämään (Beulen & Ribbers, 2002). Kwein (1998) mukaan IT –strategia voidaan nähdä liiketoimintastrategian toteuttajana, jolloin niiden tulee olla linjassa keskenään. IT –prosessien tulee tukea yritystoimintaa ja niiden tulee olla ymmärrettäviä ja elegantteja. Kwei (1998) näkeekin informaatioteknologian liiketoiminnan mahdollistajana. Hänen mukaansa IT –strategiaa tehtäessä tulee palata aina uudelleen ja uudelleen tarkastelemaan liiketoimintastrategiaa, jotta molemmat saadaan tukemaan toisiaan. Beulen & Ribbers (2002) huomauttavat, että yrityksen päättäessä ulkoistaa IT:nsa, tulee heidän kiinnittää ulkoistukseen erityistä huomiota, koska liiketoiminta– ja IT –strategian yhtenäistäminen on tällöin entistä haastavampaa. Strategiassa tulee ottaa huomioon muun muassa teknologian ja markkinoiden kehitys. Itse IT –strategian päättäminen ja johtaminen on hankkivan osapuolen johdon vastuulla, eikä sen ulkoistamista suositella. Haastatteluiden perusteella kävi ilmi, että toimittajien asiantuntijat ovat hyvin skeptisiä hankkivan osapuolen osaamisen tasosta, mitä tulee IT –strategiaan. Heidän mukaansa hankkijan strategia rajoittuu usein vain vuosittaiseen IT –budjettiin. Monimutkaisten ulkoistussuhteiden johtaminen helpottuu, kun IT –strategia on monitasoinen (palvelee eri liiketoimintayksikköjä) ja tukee liiketoimintastrategiaa. (Beulen & Ribbers, 2002)

Beulen & Ribbersin (2002) artikkelin mukaan myös tiedonhallinnan johtaminen on hankkivan osapuolen vastuulla. Iso edellytys tehokkaalle ulkoistamiselle on, että tiedonhallinnan johtaminen toimii. Tietohallintojohtajan valinnassa on tärkeää kiinnittää huomiota siihen, että hänellä on sekä liiketoiminta- että informaatioteknologiaosaamista. Tiedonhallinnasta tulee silti olla vastuussa koko johtokunnan, eikä pelkästään tietohallintojohtajan. Tiedonhallinnon onnistunut johtaminen vaikuttaa positiivisesti hankkijan ja toimittajan väliseen yhteistyöhön parantamalla kommunikaatiota näiden välillä. (Beulen & Ribbers, 2002)

6.2.2 Sopimukset ja sopimusten hallinta

Sopimusten ja asiakkuuksien hallinta (*Contract and Account Management*, myöhemmin CAM) on hyvin tärkeässä roolissa rakennettaessa suhdetta hankkijan ja toimittajan välillä (Beulen & Ribbers, 2002). Heidän tutkimuksensa mukaan CAM:in rakenteen tulee heijastaa ulkoistavan yrityksen liiketoimintaa ja organisaatiota. Sen tarkoitus on kertoa, mitä sopimuksessa oikeastaan ollaan sovittu yhteistyöstä toimittajan ja hankkijan välillä. CAMin tarkoitus onkin edesauttaa yhteistyötä ja kommunikaatiota sekä näin parantaa asiakastyytyväisyyttä. Koska CAM tulee rakentaa hankkijan liiketoimintafunktioita tukemaan, se voidaan nähdä eräänlaisena kommunikaation kiintopisteenä, jonka avulla toimittaja pääsee kiinni näihin funktioihin. Beulen & Ribbers (2002) tutkimuksessa tärkeäksi huomioksi nousi henkilöstö CAM:in ympärillä. Kyseisen henkilöstön vaihtuvuus vaikuttaa heikentävästi yhteistyön jatkuvuuteen. Beulen & Ribbers (2002) suosittelevatkin, että toimittajan puolelta sopimusten ja asiakkuuksien hallinnan tehtäviin kannattaa käyttää kokeneita johtajia.

Myös sopimuksen rakenteen tulee Beulen & Ribbers (2002) mukaan heijastaa ulkoistavan yrityksen rakennetta. Suuri osa ulkoistussopimuksista *perustuu yritysten puitesopimuksiin* (*Corporate Framerwork Agreement*, myöhemmin CFA), jossa määritellään sopimuksen yleisiä ehtoja. Puitesopimuksen alla oleva kerros on *palvelutasosopimus* (*Service Level Agreement*, myöhemmin SLA). SLA:ssa määritellään palvelulle tietyt vaatimustasot. SLA:n tulee yleensä olla joustavin osa sopimuksessa, koska vaatimustasot ovat yleensä muuttuvia eri sopimuksissa. SLA:ta mitataan erilaisilla suorituskyvyn mittareilla, joista Beulen & Ribbers (2002) esittelevät *Balanced Score Cardin* (myöhemmin BSC). BSC:llä mitataan liiketoiminnan tavoitteita ja sen tarkoitus on hillitä liiallista tekniikkaan painottunutta keskustelua. Sitä voidaan käyttää esimerkiksi jo toimittajan valinnan yhteydessä.

Kokonaisuudessaan sopimukset voivat joustavuuden suhteen hyvin erilaisia toisiinsa nähden. Osa sopimuksista tähtää keskitettyyn yhteistyöhön, kun taas toiset sopimukset eivät sido hankkijaa toimittajaan millään tavalla. Beulen & Ribbers (2002) arvioivat, että joustavuus sopimuksissa tulee lisääntymään. Miten edellä mainittu vaikuttaa laatuun, Beulen & Ribbers (2002) eivät osaa suoraan kertoa, mutta jonkinlais-

ta näyttöä on, että monimutkaisissa suhteissa joustavuutta sopimuksissa kaivataan lisää. Sopimusten tarkoitus on lisätä molempien osapuolten osallistumista ja omistautumista asialla.

6.2.3 Henkilöstöresurssien saatavuus

Beulen & Ribbers (2002) nostavat esiin kolme henkilöstöön liittyvää ongelmaa toimittajan näkökulmasta: miten pitää lahjakkaimmat ihmiset töissä, globalisaation vaikutus palveluntoimitusprosesseihin ja ulkoistajalta toimittajalle siirtyneiden henkilöiden asema. Toimittajalta ulkoistettuun yritykseen siirtyneillä työntekijöillä on tietoa molempien yritysten liiketoiminnasta, mikä voidaan katsoa eduksi molempien edustajien puolelta. Toisaalta ulkoistuksessa haetaan yleensä myös tietynlaista muutosta toimintatapoihin, jolloin näin ei välttämättä kannata toimia. Beulen & Ribbers (2002) tähdentävät henkilöstön sijoitussuunnitelman tärkeyttä, jolloin jatkuvuus, suunnitelmallisuus ja resurssit tulevaisuudessa saadaan paremmin turvattua. Lähtökohta henkilöstön sijoitussuunnitelmaan on nykyiset palvelut, arvio kyvystä täyttää nykyiset palvelut, sekä arvio tulevaisuuden tarpeista.

Beulen & Ribbers (2002) kuitenkin tuovat esille, että pelkästään parhaiden henkilöiden osallistuminen projektiin ei riitä, kun kyseessä on monimutkainen ulkoistaminen. Siihen tarvitaan kappaleessa aiemmin mainittuja IT-strategiaa, tiedonhallintaa, sopimuksia ja sopimusten hallintaa sekä näiden kaikkien yhdistämistä toimivan johdon avulla.

7 Yhteenveto

Puhuttaessa hajautetusta ohjelmistonkehityksestä on ensin syytä tutustua kirjallisuudessa käytettyihin termeihin. Hajautettu ohjelmistonkehitys, Distributed Software Development (DSD), tarkoittaa kehitystä, jossa samaa ohjelmistoa tekevä tiimi/tiimit on hajautettu maantieteellisesti eri paikkoihin. Paikat voivat olla hyvinkin lähellä toisiaan, mutta päivittäistä kasvokkain tapahtuvaa kommunikaatiota ei ole. DSD ei ota kantaa siihen, tapahtuuko kehitys yrityksen sisällä vai onko kehitys ulkoistettua. Globaali ohjelmistonkehitys, Global Software Development (GSD) on hajautettua kehitystä, jossa samaan projektiin kuuluvia ihmisiä työskentelee eri valtioissa. Myöskään GSD ei ota kantaa siihen, tapahtuuko kehitys yrityksen sisällä vai onko kehitys ulkoistettua. Offshore Software Development on terminä hyvin samankaltainen kuin GSD. Jo itse termi Global Software Outsourcing (GSO) kertoo, että kyseessä on globaalisti ulkoistettu ohjelmistonkehitys. Kehitys siis tapahtuu vähintään kahdessa eri valtiossa ja yhteistyössä on vähintään kaksi eri yritystä. Tutustuessa alan kirjallisuuteen, ei tule kumminkaan olettaa, että termejä käytetään täysin yhtenäisesti. Osassa tätä työtä varten läpikäydyissä kirjoituksissa termi GSD sisälsi sen perusoletuksen, että hajautettu kehitys tapahtui ulkoistetusti.

Sekä yrityksen sisällä että ulkoistetulla globaalilla ohjelmistonkehityksellä on hyvin samanlaisia haasteita. Ulkoistetun kehityksen erikoishaasteina on kahden eri yrityksen kulttuurien sovittaminen, mutta muuten ongelmat ovat melko samankaltaisia. Myös saman yrityksen eri toimipisteissä saattaa olla hyvinkin suuria kulttuurillisia eroja, mikäli kulttuuria ei ole ohjattu johdonmukaisesti samanlaiseksi. Suurin osa kaikista globaalin ohjelmistonkehityksen haasteista johtuu kommunikaatiosta. Kommunikaatio-ongelmiin johtavat syyt taas ovat pääasiassa kulttuurillisia ja aikaerosta sekä maantieteellisesti erosta johtuvia ongelmia. Viimeksi mainittuihin ongelmiin ratkaisuksi on ehdotettu erilaisia teknologioita, jotka mahdollistavat synkronisen ja asynkronisen kommunikaation työntekijöiden kesken. Teknologia auttaa myös työn tulosten ja tiedon jakamisessa tiimien kesken. Kulttuurien yhteentörmäys taas vaatii projektin avainhenkilöiltä ja johtajilta pelisilmää ja strategiaa, jotta suurimmilta vaikeuksilta vältytään.

Hajautettu ohjelmistonkehitys tarjoaa kumminkin hyötyjä, joita muin tavoin voi olla hankala saavuttaa. Yksi suurimmistä hyödyistä on pääsy henkilöstöresurssien pariin, joita muuten ei olisi saatavilla. Tärkein motivaatio suurimmalla osalla yrityksiä on kuitenkin mahdollisuus kustannussäästöihin muun muassa siirtämällä tuotantoa edullisimpiin valtioihin. Jonkinlaista näyttöä on myös siitä, että tuotteen saaminen markkinoille on helpompaa, kun sitä kehitetään maassa, jonka markkinoille tuote suunnataan.

Varsinkin varhaisessa vaiheessa, kun GSD:tä alettiin hyödyntämään, odotettiin hyötyjä aikaerosta. Aikaeron varaan laskettiin, että kehitystä voidaan tehdä lähes kellon ympäri, esimerkiksi siirtämällä päivällä tehty koodi tarkistettavaksi yön ajaksi. Virheiden korjauksissa aikaeroa onkin käytetty hyödyksi onnistuneesti. Kokonaisuudessaan hajautetussa ohjelmistonkehityksessä aikaeron tuomat haitat ovat kuitenkin suurempi kuin se antamat hyödyt. Jo pienikin aikaero saattaa vähentää synkronista kommunikointiaikaa useammalla tunnilla, joka taas vaikeuttaa kehitystyötä aiheuttaen mahdollisesti viivästyksiä ja laatuongelmia. Merkittävä osa tietojärjestelmien ulkoistuksessa tapahtuvista ongelmista johtuukin jollain tavalla huonosta kommunikaatiosta. Kommunikaatio-ongelmien voittamiseen tarvitaan GSD:hen sopivia prosesseja. Tehokkaat tiimit käyttävätkin GSD:ssä prosesseja, jotka on suunniteltu erityisesti synkronisen ja asynkronisen kommunikaatio-ongelmien voittamiseen. (DeLone et al. 2005).

Mishra & Mishra (2012) mukaan ohjelmistojen laadunvarmistusprosessissa tarkastus on erittäin suuressa roolissa. Tarkastuksesta saatuja tietoja voidaan käyttää laadunvarmistuksen seuraamiseen. Khanin et. al. (2013) toteavat laadunvalvonnan toimenpiteiden suunnittelun oleva erittäin tärkeää globaalissa ohjelmistonkehityksessä. Mitä varhaisemmassa vaiheessa virheet löydetään, sitä vähemmän ne tuottavat lisäkustannuksia ja ohjelmiston laatu on todennäköisesti parempaa. Caivano et. al. (2011) ja Hedberg & Harjumaa (2002) mukaan perinteiset ohjelmistojen tarkistusmetodit sopivat huonosti hajautettuun ohjelmistonkehitykseen. GSD:ssä tarkistusmetodit tulisi suunnitella sellaisiksi, että ne tukevat kommunikaatiota internetin välityksellä. Hedberg & Harjumaa (2002) kertovat virtuaalisesta ohjelmistontarkastusprosessista, jossa käytetään tähän sopivaa työkalua tarkastuksen liittyvän tiedonhallintaan.

Tiedonhallinta on muutenkin tärkeässä asemassa GSD:ssä. Tiedonhallinta on lähtökohtaisesti johdon haaste, johon on olemassa erilaisia ratkaisuja. Tiedonhallinta vaatii strategiaa, jonka perimmäisenä tarkoituksena on varmistaa relevantin tiedon helppo löytäminen, tiedon jakaminen kaikille projektin osallisille ja tiedon yhdistämisen tekeminen mahdollisimman mutkattomaksi (Desouza et. al. 2006). Kommunikaatioongelmia esiintyy kehittäjien lisäksi myös asiakkaan ja toimittajan välillä. Asiakkaan ja toimittajan välistä suhdetta voidaan parantaa muun muassa sopimusten ja sopimusten hallinnan avulla. Varsinkin ketterien menetelmien käyttö GSD:ssä tarvitsee hyvää kommunikaatiota asiakkaan ja toimittajan välillä, jotta laadukasta ohjelmistoa on mahdollista rakentaa. Kommunikaatiorikkaan ympäristön rakentaminen on tärkeä tekijä ketteriä menetelmiä hyödyntäessä (Layman et al., 2006).

Tätä pro gradua varten tutkituissa artikkeleissa ollaan hyvin yksimielisiä siitä, että hajautetun ohjelmistokehityksen myötä projektin johtamiseen kohdistuu uusia haasteita. Khan et. al. (2013) mukaan laadunvalvonnan toimenpiteiden suunnittelu ja johtaminen on hajautetussa ohjelmistonkehityksessä erittäin tärkeää. He kertovat artikkelissaan, että GSD:n näkökulmasta laadunvalvonnan johtaminen voidaan nähdä erilaisten käytäntöjen kokoelmana. Kumari et. al. (2014) mukaan ohjelmistojen laadunvarmistus koskee koko ohjelmiston kehitysprosessia. Laadunvalvonta ja ohjelmistojen testaus on lähtökohtaisesti hyvin samanlaista hajautetussa ja perinteisessä ohjelmistonkehityksessä, vaikka perinteiset tavat eivät ole suoraan siirrettäviä hajautettuun kehitykseen. Huomiota tulee kiinnittää toimenpiteiden suunnitteluun ja johtamiseen ottaen huomioon GSD:n ongelmakohdat – kommunikaation, kulttuurierot ja aikaeron. Ja kuten aiemmin todettua, itse virheiden korjaamisessa aikaerosta voi olla jopa hyötyä. GSD -projekteissa puutteiden varhainen löytäminen on tärkeää (Ivček & Galinac, 2008, Khan et. al. 2013). Ortogonaalinen virheluokittelu voi olla hajautetussa ohjelmistonkehityksessä apuna puutteiden havaitsemisessa, koska sitä käytetään sovelluskehityksessä aikaisessa vaiheessa ennaltaehkäisemään virheiden syntyä (Chillarege et. al 1992).

GSD on hyvin monimutkainen kokonaisuus. Sen tehokas hyväksikäyttäminen suurissa projekteissa vaatii laajaa näkemystä muun muassa ohjelmistonkehityksestä, laadunvalvonnasta, organisaatiosta, kulttuureista, kommunikaatiosta, sopimusten hallinnasta, testaamisesta ja tiedonhallinnasta. Esimerkiksi perinteisen vesiputousmallin ja

iteratiivisen kehittämisen yhdistäminen toimivaksi kokonaisuudeksi multikulttuurisessa ympäristössä kehittäjäryhmien ja asiakkaiden ollessa tuhansien kilometrien päässä toisistaan, kuulostaa hieman haasteelliselta, vaikka käytössä olisikin monipuolisia kommunikaatiomalleja. Mahdollista se kuitenkin on. GSD:n tarjoamat mahdolliset kustannussäästöt ja pääsy osaavien työntekijöiden pariin tulee luultavimmin lisäämään hajautettua ohjelmistonkehitystä tulevaisuudessa. Mikäli ohjelmistoista halutaan laadukkaita ja kustannussäästöjä halutaan tosissaan saavuttaa, vaatii tämä erityisen paljon kehityksen johdolta.

Viitteet

Aissaouia, N., Haouaria, M. and Hassinib, E. 2007. *Supplier selection and order lot sizing modeling: A review, Computers & Operations Research*, 34, 3516–3540.

Battin, R.D., Crocker, R., Kreidler, J., Subramanian, K. (2001): *Leveraging Resources in Global Software Development*. *IEEE Software* 18(2), 70–77 (2001)

Beulen, E., Ribbers, P., (2002): *Managing complex IT outsourcing-partnerships. In*: *System Sciences, 2002. HICSS. Proceedings ot the 35th Annual Hawaii International Conference on System Sciences.*

Bird, C., Nagappan, N., Devanbu, P., Gall, H., Murphy, B. (2009): *Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista. Communications of the ACM - A Blind Person's Interaction with Technology*, Volume 52 Issue 8, August 2009, Pages 85-93

Brannen, M.Y., Salk, J.E. (2000): Partnering across borders: *Negotiating organizational culture in a German-Japanese joint venture, Human Relations* Vol. 53, Iss. 4, pp. 451-487

Caivano, D., Lanubile, F., Visaggio, G (2001): *Scaling up Distributed Software Inspection, Proceedings of the ICSE Workshop on Software Engineering over the Internet*.

Carmel, E., Agarwal R. (2001): *Tactical Approaches for Alleviating Distance in Global Software Development. IEEE Software* Mar/Apr2001, Vol. 18 Issue 2, Pages 22-29.

Chillarege, R., Bhandari I.S., Char J.K., Halliday M.J., Moebus D.S., Ray B.K., Wong M. (1992): *Orthogonal Defect Classification-A Concept for In-Process Measurements. IEEE Transactions on Software Engineering*, 18, 11, 1992, 943-956.

Ciolkowski, M., Laitenberger, O., Biffl, S. (2003): *Software Reviews: The State of the Practise, IEEE Software,* Vol. 20, Iss. 6, pages 46-51.

Conchúir, E.Ó., Ågerfalk, P.J., Olsson H.H., Fitzgerald, P. (2009): *Global software development: Where are the benefits? Communications of the ACM, Vol. 52, Iss. 8, pages 127-131.*

Cusumano, M.A. (2008): *Managing Software Development in Globally Distributed Teams.* Communications *of the ACM Feb2008*, Vol. 51 Issue 2, pages 15-17.

Damian, D., Lanubile, F., Oppenheimer, H.L. (2003): *Addressing the challenges of software industry globalization: The workshop on global software development, Proceedings - International Conference on Software Engineering,* pages 793-794.

Damian, D., Izquierdo, L., Singer, J., Kwan, I., (2007): *Awareness in the wild: why communication breakdowns occur*. *In: International Conference on Global Software Engineering*, Pages. 81–90.

Delone, W., Espinosa, J.A., Lee, G., Carmel, E. (2005): *Bridging Global Boundaries for IS Project Success*. *In: 38th Annual Hawaii International Conference on System Sciences (HICSS 2005*) - Track 1, vol. 01, IEEE Computer Society, Los Alamitos (2005)

Desouza, K.C., Awazu. Y., Baloh, P. (2006): *Managing knowledge in global software development efforts: Issues and practices, IEEE Software* vol. 23, Iss. 5, pp. 30-37

Ebert, C., De Neve, P. (2001): *Surviving Global Software Development*. *IEEE Software* 18(2), 62–69 (2001)

Espinosa, J.A., Carmel, E. (2003): *The Effect of Time Separation on Coordination Costs in Global Software Teams: a Conceptual Foundation*. *Software Process: Improvement and Practise*, volume 8, Issue 4, Pages 249-266, 2003.

Evaristo, J.R., Scrudder, R (2000): *Geographically Distributed Project Teams: A Dimensional Analysis*. *Proceedings of the 33rd Hawaii International Conference on System Sciences* – 2000. IEEE CS Press, 2000, pp. 7052–7063.

Gopal, A., Mukhopadhyay, T., Krishnan, M.S. (2002): The Role of Software Processes and Communication in Offshore Software Development.

Grinter, R.E., Herbsleb, J., Perry, D. (1999): *The geography of coordination: dealing with distance in R&D work. Proceeding, GROUP '99 Proceedings of the international ACM SIGGROUP conference on Supporting group work.* Pages 306-315, 1999.

Gumm, D. (2006): *Distribution Dimensions in Software Development Projects: A Taxonomy*. *IEEE Software* 23(5), pages 45–51

Hedberg, H., Harjumaa, L. *(2002): Virtual software inspections for distributed software engineering projects, Proceedings of the ICSE International Workshop on Global Software Development.* Pages 17-20.

Heeks, R., Krishna, S., Nicholson, B., Sahay, S. (2001): Synching or Sinking: Global Software Outsourcing Relationships, IEEE Software, Vol. 18, Iss. 2, pp. 54-60.

Herbsleb, J.D., Grinter R.E. (1999): *Splitting the organization and integrating the code: Conway's law revisited, Proceedings - International Conference on Software Engineering*, pp. 85-96

Herbsleb, J.D., Moitra, D. (2011): *Global software development*. IEEE Sofware, vol. 18, pp 16-20.

Holmström, H., Conchúir, E., Ågerfalk, P.J., Fitzgerald, B. (2006): *Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance*, *International Conference on Global Software Engineering* (ICGSE2006), Costão do Santinho, Florianópolis, Brazil, October 16-19 2006.

IEEE standard classification for software anomalies. *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, (2010)

Ivček, M., Galinac, T. (2008): *Aspects of quality assurance in global software development organization. MIPRO 2008 – 31st International Convention Proceedings: Telecommunication and Information* pp. 150-155

Khan, K., Khan, A., Aamir, M., Khan, M.N.A. (2013): *Quality Assurance Assessment in Global Software Development. World Applied Sciences Journal* 24 (11): 1449-1454, 2013

Khan, S.U., Niazi, M., Ahmad, R. (2010): *Factors Influencing Clients in the Selection of Offshore Software Outsourcing Vendors: An Exploratory Study Using a Systematic Literature Review. The Journal of Systems and Software* 84, 686-699.

Kitchenham, B., Pfleeger. S.L. (1996): *Software Quality: The Elusive Target. IEEE Software* Jan96, Vol. 13 Issue 1, p12.

Korkala, M., Pikkarainen, M., Conboy, K. (2010): *A Case Study of Customer Communication in Globally Distributed Software Product Development. Proceedings of the 11th International Conference on Product Focused Software*, pp 43-46, 2010.

Krishna, S., Sahay, S., Walsham, G. (2004): *Managing Cross-Cultural issues In Global Software Outsourcing, Communications of the ACM* 2004, Vol. 47. No. 4, pages 62-66.

Kumari, R., Chhabra, P., Gogia, R. (2014): *Software Quality Assurance, International Journal of Research*, Vol. 1, Issue 10, pages 12-16.

Kwei, R. (1998): *Aligning business and IT strategy, Health Management Technology,* Vol. 19, Issue 2, pages 72-74.

Layman, L., Williams, L., Damian, D., Bures, H. (2006): *Essential communication practices for Extreme Programming in a global software development team. Information and Software Technology*, Vol. 48, Iss. 9, pp. 781-794.

Mishra, D., Mishra, A. (2012): *A Global Software Inspection Process for Distributed Software Development, Journal of Universal Computer Science*, Vol. 18, Iss. 19, pages 2731-2746.

Nagappan, N., Murphy, B., Basili, V. (2008): *The influence of organizational structure on software quality: An empirical case study. In ICSE ´08: Proceedings of the 30th international conference on Software engineering*, pages 521–530, Leipzig, Germany, 2008.

Nicholls, D., Lein, P., McGibbon, T. (2011): *Achieving System Reliability Growth Through Robust Design and Test.* RIAC 2011, pages 164-169.

Nisar, MF., Hameed, T. (2004): *Agile methods handling offshore software development issues, INMIC 2004: 8th International Multitopic Conference, Proceedings,* pages 417-422.

Paasivaara, M., Lassenius, C. (2006): *Could Global Software Development Benefit from Agile Methods? IEEE International Conference on Global Software Engineering (ICGSE'06)*, pages 109-113.

Ramesh, B., Cao, L., Mohan, K., Xu, P. (2006): *Can distributed software development be agile? Communications of the ACM*, 49, 41-46.

Raninen, A., Toroi, T., Vainio, H., Ahonen, J.J. (2012): *Defect Data Analysis as Input for Software Process Improvement. Lecture Notes in Computer Science* Volume 7343, 2012, pp 3-16

Sangwan, R.S., Ros, J. (2008). *Architecture leadership and management in globally distributed software development. Proceedings - International Conference on Software Engineering* pp. 17-21, 2008

Stein, M., Riedl, J., Harner, SJ., Mashayekhi, V. (1997): *A case study of distributed, asynchronous software inspection, Proceedings of the 1997 International Conference on Software Engineering,* pages 107-117.

Sullivan, M., Chillarege, R. (1991): *Software Defects and their Impact on System Availability - A Study of Field Failures in Operating Systems. Fault-Tolerant Computing, 1991*. FTCS-21. Digest of Papers. Twenty-First International Symposium.

Toroi, T., Raninen A., Vainio H. (2012): *Using Functional Defect Analysis as an Input for Software Process Improvement: Initial Results. Communications in Computer and Information Science* Volume 301 CCIS, 2012, pages 181-192

Toroi, T., Raninen A., Vainio H., Väätäinen, L. (2013): *Identifying Process Problems with the SAWO Functional Defect Classification Scheme. Systems, Software and Services Process Improvement, Communications in Computer and Information Science* Vol. 364, pp 72-83

ul Haq, S., Raza, M., Zia, A., Khan, M.N.A. (2011): *Issues in Global Software Development: A Critical Review. J. Software Engineering & Applications*, 2011, 4, 590-595

Wagner, S. (2006): *A literature survey of the quality economics of defect-detection techniques. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, Pages 194-203.

Wagner S. (2008): Defect Classification and Defect Types Revisited (2008) *DEFECTS '08 Proceedings of the 2008 workshop on Defects in large software systems*, ACM New York, 39-40.

Ågerfalk, P., Fitzgerald, B., Holmstrom, H., Ó Conchúir, E. (2008): *Benefits of Global Software Development: The Known and Unknown*, *in Q. Wang, D Pfahl, and D.M. Raffo (Eds.): Making Globally Distributed Software a Success Story, ICSP* 2008, LNCS 5007, pp. 1-9, Springer-Verlag Berlin Heidelberg.

Ågerfalk, P.J (2004): *Investigating actability dimensions: a language/action perspective on criteria for information systems evaluation. Interacting with Computers* Vol. 16 Issue 5, p957-988

http://www.agilemanifesto.org/. Haettu Internetistä 11.11.2014